CGIプログラミング

Shishir Gundavaram 著

田辺　茂也　監訳

株式会社 エディックス　訳

O'REILLY™

概要
ソケットとは
Perlのソケット I/O
ソケットライブラリ
ハイパーテキスト（HTTP）リンクをチェックする
Archie
Web上のネットワークニュース
マジッククッキー（Magic Cookies）
サーバを使用して状態を維持管理する
チャイルドプロセスのフォーク/生成

10章 インターネット情報サーバへのゲートウェイ

10.1 概 要

これまで、Archie（これはFTPサイトを検索するもの）やNNTP（Usenetニュースサーバ）といった、インターネット上の情報サーバについて聞いたことがあるでしょう。Web同様、これらのサービスはTCP/IPの最上位のプロトコルとして動作します。これらのサービスをWeb上で利用できるようにするために、TCP/IPネットワークプロトコルを使用して、他のインターネット情報サーバへのクライアントとして動作するCGIアプリケーションを開発することができます。

それでは、サーバがどのように機能するかについてから説明を始めましょう。電子メールアプリケーションを取り上げます（この理論は他のサーバにも適用することができます）。ほとんどのメールプログラムは、典型的には/var/spool/mailディレクトリにある特定ファイルにユーザのメッセージをセーブします。違うホスト上の誰かにメールを送る場合、メールプログラムは、そのマシン上にある受取人のメールファイルを見つけ出し、そこにメッセージを追加しなければなりません。しかし、リモートホスト上で直接ファイルを操作することはできないのに、どのようにしてメールプログラムはこのタスクを正確に実行するのでしょうか。

この疑問に対する答えは、interprocess communication（IPC）です。リモートホスト上のプロセスは、そのマシン上のメールプロセス用メッセンジャーとして動作します。ローカルプロセスは、メールを配信するために、ネットワークを介してこのリモートエージェントと通信します。そのため、リモートプロセスはサーバと呼ばれ（このプロセスは発行したリクエストをサービスします）、ローカルプロセスはクライアントと呼ばれます。Webは同じ原理で動作します。ブラウザは、リクエストを解釈して実行するHTTPサーバにリクエストを発行するクライアントです。

ここで覚えておくべき一番重要なことは、クライアントとサーバは同じメッセージを伝える必要が

あるという点です。言い換えると、特定のクライアントは指定したサーバと動作するように設計されています。例えば、Archieクライアントは Webサーバとは通信することはできません。しかし、サーバのデータストリームと、出力として生成されるストリームがわかっていれば、「9章　ゲートウェイ、データベース、検索/索引ユーティリティ」で紹介したように、それと通信するCGIプログラムを記述することができます。

本章で紹介する非常に便利なアプリケーションは、クライアントとサーバの両方を作成するアプリケーションです。これはクッキーハンドラとなるもので、データが複数フォームに入力される場合に、そのデータを記録し続けることができます。

この通信プロトコルはUNIXシステムの種類によって変わってきます。System Vと呼ばれているAT & TのUNIXバージョンでは、ネットワークを介してプロセスと通信するためのSTREAMSを提供しています。一方、カリフォルニア大学バークレー校のBSD UNIXでは、ソケットと呼ばれるネットワーク通信用オブジェクトを実装しています。本章では、(PC業界も採用した) BSDソケットについてのみ取り上げます。これはネットワーク通信をハンドルするための一番普及している方法です。

10.2　ソケットとは

ほとんどの企業では、電話のやりとりを行うゲートウェイとして機能する電話交換機を備えているでしょう。ソケットは電話交換機にたとえることができます。リモートホストに接続したければ、最初に通信を行うためのソケットを作成する必要があります。これは"9"をダイヤルして、交換機を通じて外線につなぐことに似ています。

同様に、リモート(もしくはローカル)ホストからの接続を受け付けるサーバを作成したければ、定期的に接続を待ち受けるためのソケットを設定する必要があります。ソケットはホストのIPアドレスとそれが待ち受けるポートによって、インターネット上で識別されています。接続がいったん確立されると、新しいソケットが作成され、この接続をハンドルします。そして始めのソケットは元に戻って他の接続を待ち受けます。交換機はこれと同様に動作しています。外線呼び出しをハンドルし、それを適切な内線にルートして、再度戻って他の呼び出しを受け付けます。

10.3　PerlのソケットI/O

Perlでソケットを設定するときに使用する関数は、それに対応するUNIXシステム関数と同じ名前になっています。ただし、ソケット関数の引数は若干異なっています。それでは、fingerサーバにクライアントを実装する事例について見ていきましょう。

これはCGIスクリプトではありません。しかし、これをCGIスクリプトに変換するのは非常に簡単です。これはコマンド行から実行して1つの引数、ユーザ名を渡すようになっています。

```
% finger_client username[@host]
```

見ればわかるように、呼び出しフォーマットはUNIX fingerコマンドのフォーマットと同じです。実際には、このプログラムはこれとまったく同様に動作します。

```
#!/usr/local/bin/perl

require "sys/socket.ph";
```

Perlヘッダファイル"socket.ph"にはソケットのさまざまなタイプに関する定義、アドレシング技法などが入っています。これらの定義について少し見ていきましょう。

このファイルが見つからないと、すべてのC/C++ヘッダファイルをPerlが理解できるフォーマットに変換するためのh2ph Perlスクリプトを実行する必要があります。それでは、プログラムを続けましょう。

```
chop ($hostname = `/bin/hostname`);
$input = shift (@ARGV);
```

現在のホスト名はUNIX hostnameコマンドを使用して取り出されます。また、スクリプトへの入力はinput変数に格納されます。shiftステートメントは配列の先頭要素を返します。

```
($username, $remote_host) = split (/@/, $input, 2);
```

指定したユーザ名とリモートホストがinput変数から分割されます。

```
unless ($remote_host) {
    $remote_host = $hostname;
}
```

ホストを指定しないと、ローカルホストがデフォルトになります。

```
$service = "finger";
```

いったんソケットを作成すると、通常ではこれはマシン上のポートにアタッチされます。メッセージまたはリクエストをサーバに送るには、サーバが稼働しているポートにそれを送る必要があります。通常、共通サーバのほとんど（FTP、Archie、Gopher、HTTP、Fingerなど）は、特定のポート上で稼働していて、ネットワークを介したほとんどすべてのホストでも同じです。そうでなければ、異なるマシンのクライアントはサーバにアクセスできなくなってしまいます。なぜなら、

サーバがつながっているポートがわからなくなるからです。アタッチされているサーバとポートの全リストは、/etc/servicesファイルにリストアップされています。

この場合、ポート番号ではなくサーバ名を指定しています。興味があれば、fingerサーバはポート79で実行しています。後で説明しますが、getservbyname関数はサービス"finger"を正しいポート番号に変換します。

```
$socket_template = "S n a4 x8";
```

これは16バイト構造を表していて、インターネット上で内部プロセス通信を行うために、ソケットと一緒に使用されます。この最初の2バイトは、インターネットアドレスファミリ用数値コードを、ローカルマシンがshort型整数を使用する場合のバイト順で表しています。次の2バイトは、インターネットの標準バイト順（例えば、その整数の上位バイトが一番左のバイトに格納され、一方下位バイトは一番右のバイトに格納されるというビッグエンディアン）を使用して接続するポート番号を表しています。4バイト目から8バイト目まではIPアドレスを表しています。最後の8バイトには"\0"キャラクタが入っています。実行してみればすぐわかります。

```
$tcp = (getprotobyname("tcp"))[2];
```

fingerサーバはTCPプロトコル（この内容については気にする必要はありません）として設定されているので、このプロトコルを識別するための数値コードを獲得する必要があります。getprotobyname関数は、名前、エイリアス、そして指定したプロトコルの番号を返します。この場合には、3番めの要素だけを格納しています。他は不要です。余談ですが、getprotobyname関数を呼び出す代わりに、（sockets.phヘッダファイルにある）定数AF_NSを使用することもできます。

```
if ($service !~ /^\d+$/) {
    $service = (getservbyname ($service, "tcp"))[2];
}
```

変数に指定されたサービスが数値でなければ、getservbyname関数は/etc/servicesファイルを使用してポート番号を検索します。

```
$current_address = (gethostbyname ($hostname))[4];
$remote_address  = (gethostbyname ($remote_host))[4];
```

gethostbyname関数は、ネットワーク上の位置を表しているパックされた文字列にホスト名を変換します。このパックされた文字列は公分母のようなもので、多くの関数に渡す必要があります。この文字列をIPアドレスに変換したければ、アンパックする必要があります。

```
@ip_numbers = unpack ("C4", $current_address);
$ip_address = join (".", @ip_numbers);

unless ($remote_address) {
    die "Unknown host: ", $remote_host, "\n";
}
```

リモートホストを表しているパックされた文字列が定義されていないと、これはその位置が存在しないということを意味しています。

```
$current_port = pack ($socket_template, &AF_INET, 0, $current_address);
$remote_port  = pack ($socket_template, &AF_INET, $service, $remote_address);
```

この2つの行は非常に重要です。前述したソケットテンプレートを使用して、インターネットアドレシング方式を表している3つの値、ポート番号、そしてホスト名がパックされ、実際にソケットを作成するときに使用するソケット構造体を作成します。&AF_INETは、インターネットアドレシング（例えば、128.197.27.7）メソッドを参照するソケットヘッダファイルに定義されているサブルーチンです。&AF_UNIXなどの別のアドレシング方式をソケット用に定義することもできます。&AF_UNIXではUNIXパス名を使用して、特定ホストにローカルになっているソケットを識別します。

```
socket (FINGER, &AF_INET, &SOCK_STREAM, $tcp) || die "Cannot create socket.\n";
```

socket関数は、FINGERというTCP/IP（インターネットプロトコル）ソケットを作成します。FINGERはファイルハンドルとして実際に使用することができます。これはソケットの単純の美の1つです。複雑な接続タスクが完了すると、ファイルのように読み書きすることができるようになります。

&SOCK_STREAM（ヘッダファイルに定義されているもう1つのサブルーチン）の値は、キャラクタのストリームとしてソケットを渡り歩くデータを表しています。また、データがブロックまたはデータグラムを渡り歩く場合の&SOCK_DGRAMパラダイムを選択することも可能です。ただし、SOCK_STREAMソケットが一番使いやすくなっています。

```
bind (FINGER, $current_port)   || die "Cannot bind to port.\n";
connect (FINGER, $remote_port) || die "Cannot connect to remote port.\n";
```

bindステートメントは、現在のアドレスとポートにFINGERソケットをアタッチします。最後に、connect関数がremote_portで指定されたポートとアドレスに位置しているサーバにソケットを接続します。これらの関数のどれかがエラーになると、スクリプトは終了します。

```
$current_handle = select (FINGER);
$| = 1;
select ($current_handle);
```

このステートメントグループはソケットのバッファリングをオフにするのに使用され、ソケットを出入りするデータが正しい順序で表示されます。

```
print FINGER $username, "\n";
```

指定したユーザ名がソケットに送られます。fingerサーバはユーザ名だけを待っています。telnetを使用して（サーバがある）ポート79に接続して、fingerサーバの動作をテストすることができます。

```
% telnet acs.bu.edu 79
Trying 128.197.152.10 ...
Connected to acs.bu.edu.
Escape character is '^]'.
shishir
.
.
. (information returned by the server for user "shishir")
.
.
```

プログラムを完成させましょう。

```
while (<FINGER>) {
    print;
}
close (FINGER);
exit (0);
```

whileループは、サーバが出力した情報を読み込んで、それを表示します。ソケットから読み込むことは、（ネットワークエラーが発生する可能性があることを除けば）ファイルやパイプから読み込むのと同じです。最後にソケットがクローズされます。

このソケット作成の説明がわからなくても大丈夫です。このようなコードを記述する必要はありません。より簡単な関数セットについて手短に説明しましょう。

10.4　ソケットライブラリ

クライアントとサーバの作成作業をより簡単に行うために、さまざまなソケットやネットワーク

情報関数を1つにまとめたソケットライブラリが開発されました。以下に、このライブラリを使用した同じfingerクライアントを紹介します。

```
#!/usr/local/bin/perl

require "sockets.pl";

$service = "finger";
chop ($hostname = `/bin/hostname`);

$input = shift (@ARGV);
($username, $remote_host) = split (/@/, $input, 2);

unless ($remote_host) {
        $remote_host = $hostname;
}
```

このコードのほとんどは、1つの例外を除けば、前述した例で使用したコードと同じものです。requireコマンドでsockets.plライブラリを含ませます。

```
&open_connection (FINGER, $remote_host, $service)
    || die "Cannot open connection to: $remote_host", "\n";
```

open_connectionライブラリサブルーチンは、以下のタスクを実行します。

- リモートホストがIP番号（128.197.152.10）かIP名（acs.bu.edu）かどうかをチェックし、バックされたアドレス文字列に対して適切な変換を実行します。
- ソケットを作成します。
- ソケットを現在のホストにバインドします。
- ソケットをリモートアドレスとポートに接続します。
- ソケットのバッファリングをオフにします。

それでは、残りのプログラムを以下に示します。

```
print FINGER $username, "\n";

while (<FINGER>) {
    print;
}

&close_connection (FINGER);
exit (0);
```

close_connectionサブルーチンはソケットをフラッシュして、ソケットに残っているすべての情報をリリースし、ソケットをクローズします。見ればわかるように、このライブラリは、ネットワー

クサーバとの通信プロセス全体を容易にしてくれます。それでは、HTTPサーバとやりとりする簡単な例について見ていきましょう。

10.5　ハイパーテキスト（HTTP）リンクをチェックする

「7章　アドバンストフォームアプリケーション」のゲストブック事例についてもう一度見てみると、フィールドの1つがユーザのHTTPサーバを問い合わせていることに気付くでしょう。その時点では、ユーザが指定したアドレスが有効であるかどうかチェックするためのメソッドについては説明しませんでした。しかし、この新しいソケットとネットワーク通信の知識を使用すれば、アドレスの正当性を判断することができます。結局、Webサーバは同じインターネットプロトコルを使用する必要があります。マジックなどは使用しません。WebサーバへのTCP/IPソケット接続をオープンすると、fingerデーモン（サーバ）にコマンドを渡すのと同様に、それが認識するコマンドを渡すことができます。先に進む前に、ユーザが指定したURLを出力するゲストブックのコードの一部を以下に紹介します。

```
        if ($FORM{'www'}) {
            print GUESTBOOK <<End_of_Web_Address;

<P>
$FORM{'name'} can also be reached at:
<A HREF="$FORM{'www'}">$FORM{'www'}</A>

End_of_Web_Address

        }
```

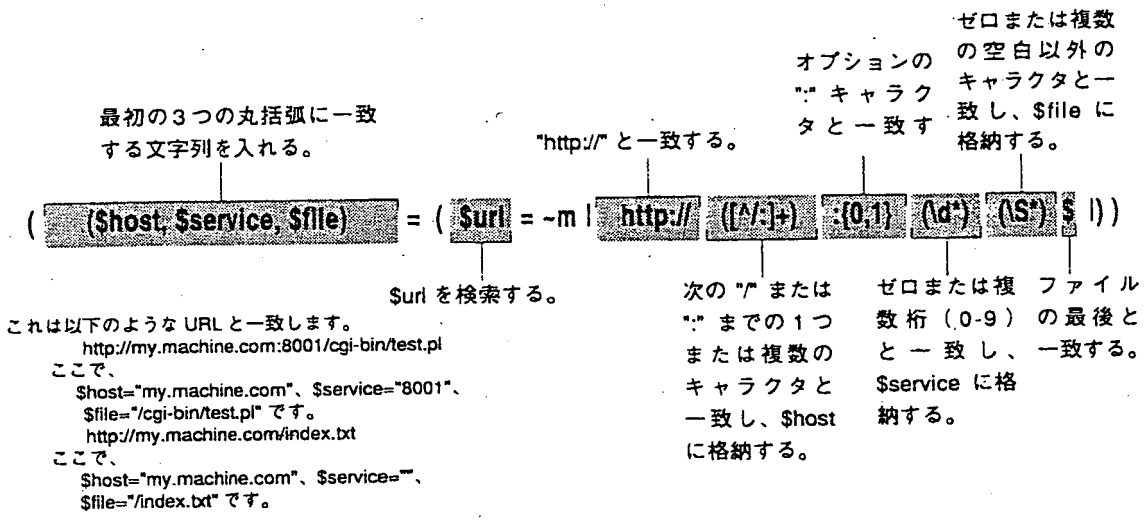
これは、有効なURLアドレスをチェックするためのソケットライブラリを利用したサブルーチンです。これは1つの引数、チェックするURLを取ります。

```
sub check_url
{
    local ($url) = @_;
    local ($current_host, $host, $service, $file, $first_line);

    if (($host, $service, $file) =
        ($url =~ m|http://([^/:]+):{0,1}(\d*)(\S*)$|)) {
```

この正規表現は指定したURLを解析して、ホスト名、（もし含まれていれば）ポート番号、ファイルを検索します。

それでは、プログラムを続けましょう。




```
chop ($current_host = `/bin/hostname`);

$host = $current_host  if ($host eq "localhost");
$service = "http"      unless ($service);
$file = "/"            unless ($file);
```

ホスト名が“localhost”として与えられると、現在のホスト名が使用されます。さらに、これらのフィールドに指定された情報がなければ、サービス名とファイルはそれぞれ“http”と“/”に設定されます。

```
&open_connection (HTTP, $host, $service) || return (0);
print HTTP "HEAD $file HTTP/1.0", "\n\n";
```

ソケットが作成され、リモートホストに対して接続しようとします。失敗すると、ゼロのエラーステータスが返されます。成功すれば、HTTPサーバにHEADコマンドが発行されます。指定したドキュメントが存在すれば、サーバは次のようなものを返します。

```
HTTP/1.0 200 OK
Date: Fri Nov  3 06:09:17 1995 GMT
Server: NCSA/1.4.2
MIME-version: 1.0
Content-type: text/html
Last-modified: Sat Feb  4 17:56:33 1995 GMT
Content-length: 486
```

ここで注目するところは先頭行です。ここにはステータスコードが入っています。ステータスコードが200なら、成功ステータスが返されます。ドキュメントがプロテクトされていたり存在しなかったりすると、それぞれ401と404のエラーコードが返されます（「3章　CGIからの出力」を参照）。

以下に、ステータスをチェックするためのコードを示します。

```
        chop ($first_line = <HTTP>);

        if ($first_line =~ /200/) {
            return (1);
        } else {
            return (0);
        }

        close (HTTP);
    } else {
        return (0);
    }
}
```

以下に、ゲストブックでこのサブルーチンをどのように使用するかを示します。

```
        if ($FORM{'www'}) {
            &check_url ($FORM{'www'}) ||
                &return_error (500, "Guestbook File Error",
                "The specified URL does not exist. Please enter a valid URL.");

            print GUESTBOOK <<End_of_Web_Address;

<P>
$FORM{'name'} can also be reached at:
<A HREF="$FORM{'www'}">$FORM{'www'}</A>

End_of_Web_Address

        }
```

それでは、既存のクライアントソフトウェアを使用して、Archieサーバへのゲートウェイを作成する事例について見ていきましょう。

10.6 Archie

Archieは、世界中の膨大なFTPサイト（とそのコンテンツ）のデータベース/インデックスです。Archieクライアントを使用すると、指定したファイルに関してデータベースを検索することができます。この事例では、Brendan Kehoe氏のArchieクライアントソフトウェア（バージョン1.3）を使用してArchieサーバに接続し、ユーザが指定した情報を検索します。ソケットライブラリを使用すれば、クライアントを簡単に記述することができますが、すばらしいものがすでに存在しているので時間の無駄でしょう。このArchieゲートウェイはMartijn Koster氏が開発したもので、

ArchiPlexをベースにしています。

```
#!/usr/local/bin/perl

$webmaster = "Shishir Gundavaram (shishir\@bu\.edu)";
$archie = "/usr/local/bin/archie";
$error = "CGI Archie Gateway Error";
$default_server = "archie.rutgers.edu";
$timeout_value = 180;
```

archie変数にはArchieクライアントへのフルパスが入っています。自分のローカルマシン上にこのパス名を使用したArchieクライアントがあることを確認してください。クライアントがなければ、クライアントがあるマシンにtelnetして、そこでこのプログラムを実行する必要があります。

検索するデフォルトサーバが格納されます。これは、ユーザがサーバの選択に失敗した場合に使用されます。

最後に、timeout_valueには、ゲートウェイがエラーメッセージを返して終了するまでの秒数が入っています。これにより、ユーザは検索結果をいつまでも待つ必要がなくなります。

```
%servers = ( 'ANS Net (New York, USA)',          'archie.ans.net',
             'Australia',                        'archie.au',
             'Canada',                           'archie.mcgill.ca',
             'Finland/Mainland Europe',          'archie.funet.fi',
             'Germany',                          'archie.th-darmstadt.de',
             'Great Britain/Ireland',            'archie.doc.ac.ac.uk',
             'Internic Net (New York, USA)',     'ds.internic.net',
             'Israel',                           'archie.ac:il',
             'Japan',                            'archie.wide.ad.jp',
             'Korea',                            'archie.kr',
             'New Zealand',                      'archie.nz',
             'Rutgers University (NJ, USA)',     'archie.rutgers.edu',
             'Spain',                            'archie.rediris.es',
             'Sweden',                           'archie.luth.se',
             'SURANet (Maryland, USA)',          'archie.sura.net',
             'Switzerland',                      'archie.switch.ch',
             'Taiwan',                           'archie.ncu.edu.tw',
             'University of Nebrasksa (USA)',    'archie.unl.edu' );
```

Archieサーバとその IP 名のいくつかは連想配列に格納されています。この配列に入っているすべてのサーバをリストアップしながら、ダイナミックにこのゲートウェイ用フォームを作成していきます。

```
$request_method = $ENV{'REQUEST_METHOD'};

if ($request_method eq "GET") {
    &display_form ();
```

このプログラムがブラウザを使用してアクセスされると、このフォームが作成され表示されます。

```
} elsif ($request_method eq "POST") {
    &parse_form_data (*FORM);
    $command = &parse_archie_fields ();
```

すべてのフォームデータがデコードされ、FORM連想配列に格納されます。parse_archie_fields サブルーチンはフォームデータを使用してクエリーを構成し、Archieクライアントに渡されるようにします。

```
    $SIG{'ALRM'} = "time_to_exit";
    alarm ($timeout_value);
```

この配列がどのように使用されるのかを理解するには、割り込みやブレークがプログラムに到着するたびにUNIXカーネルがチェックすることを理解する必要があります。何のルーチンを呼び出す必要があるのでしょうか。プログラムが必要なルーチンはシグナルハンドラです。PerlはSIG連想配列のシグナルとハンドラを関連付けます。

前述したように、タイムアウトを実装する従来の方法は、指定した秒数の後に呼び出されるようにALRMシグナルを設定する方法です。先頭行では、アラームが信号を出した際に、time_to_exit サブルーチンが実行されるように指定しています。2行めのPerl alarmコールは、$timeout_value 変数で指定した秒数で送られるようにALRMシグナルをスケジュールします。

```
    open (ARCHIE, "$archie $command |");
    $first_line = <ARCHIE>;
```

Archieクライアントに対してパイプがオープンされます。command変数には、検索する文字列はもちろんのこと、検索タイプ、Archieサーバアドレスなどのさまざまなコマンド行オプションを指定するためのクエリーが入っています。command変数はシェルに転送されるので、parse_archie_fields サブルーチンが、シェルメタキャラクタが指定されていないことを確認します。

```
    if ($first_line =~ /(failed|Usage|WARNING|Timed)/) {
        &return_error (500, $error,
            "The archie client encountered a bad request.");
    } elsif ($first_line =~ /No [Mm]atches/) {
        &return_error (500, $error,
        "There were no matches for <B>$FORM{'query'}</B>.");
    }
```

Archieサーバからの返事の先頭行に、エラーもしくは"No Matches"文字列のどちらかが入っていると、return_errorサブルーチンが呼び出されて、よりフレンドリなメッセージを返します。エ

ラーがなければ、先頭行は通常空白になります。

```
print "Content-type: text/html", "\n\n";
print "<HTML>", "\n";
print "<HEAD><TITLE>", "CGI Archie Gateway", "</TITLE></HEAD>", "\n";
print "<BODY>", "\n";
print "<H1>", "Archie search for: ", $FORM{'query'}, "</H1>", "\n";
print "<HR>", "<PRE>", "\n";
```

ヘッダ情報の通常タイプが出力されます。以下のコード行はArchieサーバからの出力を解析して、一致したファイルへのハイパーテキストリンクを作成します。

これはArchieサーバ出力の典型的なフォーマットです。必要なファイル（この場合、emacs）が見つかったところの各ホストをリストアップし、その後にその名前のファイルが入っているすべてのアクセス可能なディレクトリリストが続いています。ファイルはロングフォーマットでリストアップされていて、ファイルの日付やサイズを確認することができます。

```
Host amadeus.ireq-robot.hydro.qc.ca

    Location: /pub
      DIRECTORY drwxr-xr-x        512  Dec 18 1990  emacs

Host anubis.ac.hmc.edu

    Location: /pub
      DIRECTORY drwxr-xr-x        512  Dec  6 1994  emacs
    Location: /pub/emacs/packages/ffap
      DIRECTORY drwxr-xr-x        512  Apr  5 02:05  emacs
    Location: /pub/perl/dist
      DIRECTORY drwxr-xr-x        512  Aug 16 1994  emacs
    Location: /pub/perl/scripts/text-processing
           FILE -rwxrwxrwx         16  Feb 25 1994  emacs
```

ハイパーテキストリンクに加えることにより、この出力を拡張することができます。このように、ユーザはボタンクリックを使用して任意のホストへの接続をオープンして、ファイルを検索することができます。この出力を解析するためのコードを以下に示します。

```
while (<ARCHIE>) {
    if ( ($host) = /^Host (\S+)$/ ) {
        $host_url = join ("", "ftp://", $host);
        s|$host|<A HREF="$host_url">$host</A>|;

        <ARCHIE>;
```

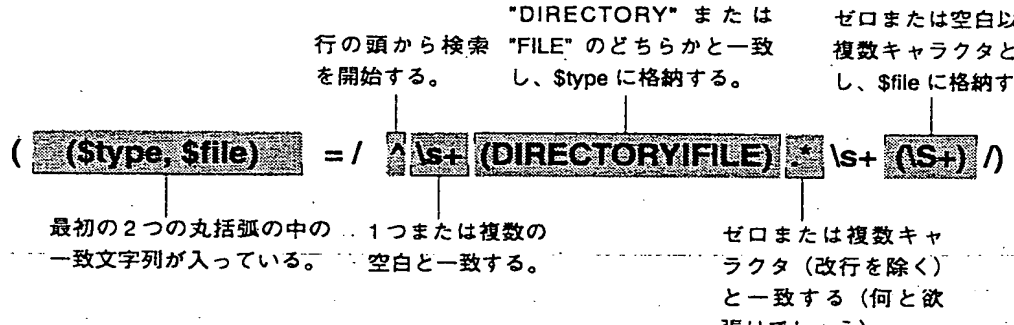
この行が"Host"で始まっていると、指定したホストが格納されます。ftp方式とホスト名を使用して、join関数によりこのホストへのURLが作成されます。例えば、ホスト名がftp.ora.comなら、

URLはftp://ftp.ora.comとなります。最後に、この行の後の空白行は破棄されます。

```
        } elsif (/^\s+Location:\s+(\S+)$/) {
            $location = $1;
            s|$location|<A HREF="${host_url}${location}">$location</A>|;
        } elsif ( ($type, $file) = /^\s+(DIRECTORY|FILE).*\s+(\S+)/) {
            s|$type|<I>$type</I>|;
            s|$file|<A HREF="${host_url}${location}/${file}">$file</A>|;
        } elsif (/^\s*$/) {
            print "<HR>";
        }

        print;
    }
```

ここでは、正規表現の特徴の1つが紹介されています。これは欲張りで、可能な限り多くのテキストを受け入れます。式(DIRECTORY|FILE).*\s+は、DIRECTORYまたはFILEと一致し、空白になるまでの多くのキャラクタと一致します。行の中にはかなりの空白がありますが、.* は最後の空白に至るまですべてを取り出します。これは、"emacs"という単語を残して最後の丸括弧に入った式(\S+)と一致します。



この式は以下のような文字列と一致します。
　DIRECTORY drwxr-xr-x 512 Jan 15 1996 emacs
ここで、$type="DIRECTORY"$file="emacs" です。

　FILE drwxr--r--　　16 Jan 16 1996 emacs.1
ここで、 $type="FILE" and $file="emacs.1" です。

残りの行は同じ方法で読み込まれて解析され、表示されます（図10–1参照）。行が空だと、各エントリーの最後を示すために水平の罫線が出力されます。

```
    $SIG{'ALRM'} = "DEFAULT";
    close (ARCHIE);

    print "</PRE>";
    print "</BODY></HTML>","\n";
```

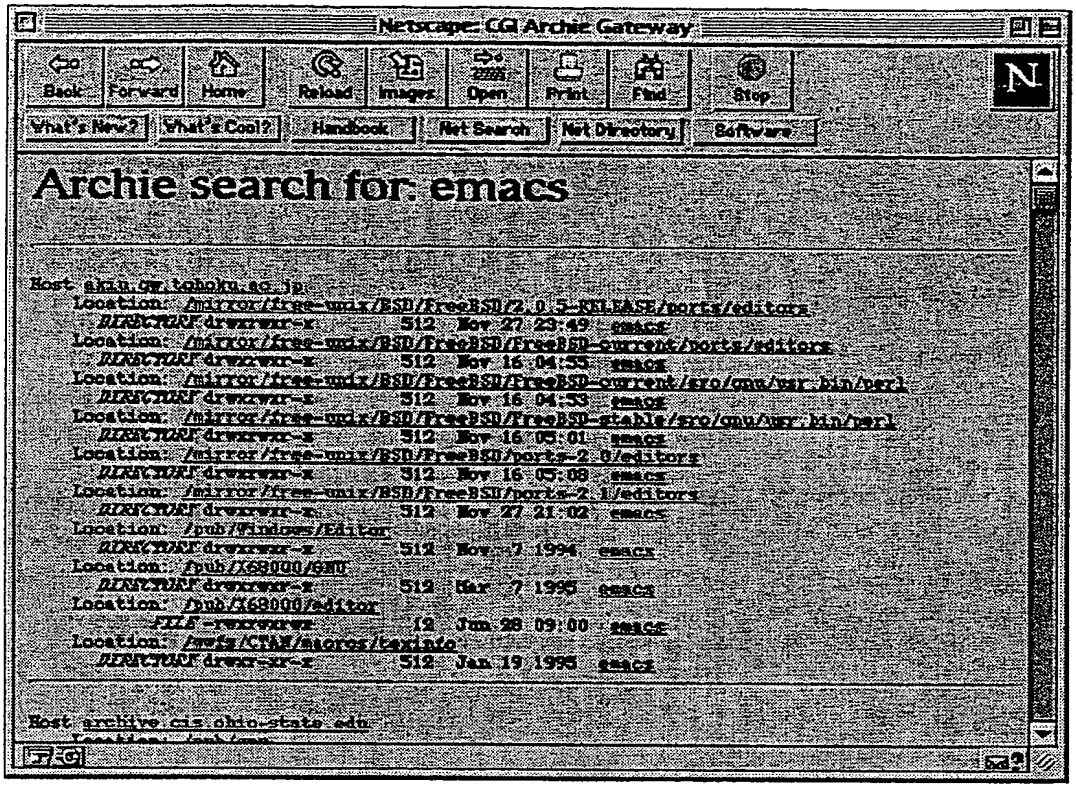


図10–1 Archie結果

最後に、ALRMシグナルがリセットされ、ファイルハンドルがクローズされます。

```
} else {
    &return_error (500, $error, "Server uses unspecified method");
}

exit (0);
```

シグナルがtime_to_exitサブルーチンを実行するには、どのようにしてSIG配列を設定するか、覚えていますか。これを以下に示します。

```
sub time_to_exit
{
    close (ARCHIE);

    &return_error (500, $error,
        "The search was terminated after $timeout_value seconds.");
}
```

このサブルーチンを実行すると、検索可能な180秒が渡されて、それがスクリプトを終了する時

間となります。一般的には、Archieサーバは一致したFTPサイトとそのファイルを迅速に返しますが、リクエストが順番待ちになる場合があります。そのような場合には、いつまでもユーザを待たせるよりも、スクリプトを終了したほうが賢明です。

ここで、parse_archie_fieldsサブルーチンを使用して、Archieクライアントが認識するコマンドを作成する必要があります。

```
sub parse_archie_fields
{
    local ($query, $server, $type, $address, $status, $options);

    $status = 1;

    $query = $FORM{'query'};
    $server = $FORM{'server'};
    $type = $FORM{'type'};

    if ($query !~ /^\w+$/) {
        &return_error (500, $error,
            "Search query contains invalid characters.");
```

queryフィールドに英数字以外のキャラクタ（A–Z、a–z、0–9、_以外のキャラクタ）が入っていると、エラーメッセージが出力されます。

```
    } else {
        foreach $address (keys %servers) {
            if ($server eq $address) {
                $server = $servers{$address};
                $status = 0;
            }
        }
```

foreachループはservers連想配列のキーの中を繰り返します。ユーザが指定したサーバが配列に入っている名前と一致すると、そのIP名がserver変数に格納され、ステータスはゼロに設定されます。

```
    if ($status) {
        &return_error (500, $error, "Please select a valid archie host.");
```

ゼロ以外のステータスは、ユーザがArchieサーバに対して無効アドレスを指定したことを表しています。

```
    } else {
        if ($type eq "cs_sub") {
            $type = "-c";
        } elsif ($type eq "ci_sub") {
```

```
            $type = "-s";
        } else {
            $type = "-e";
        }
```

ユーザが“Case Sensitive Substring”を選択すると、“-c”スイッチが使用されます。“-s”スイッチは“Case Insensitive Substring”を表しています。ユーザがオプションを選択しなかった場合には、“-e”スイッチ（“Exact Match”）が使用されます。

```
            $options = "-h $server $type $query";

            return ($options);
        }
    }
}
```

すべてのオプションが入っている文字列が作成され、メインプログラムに戻ります。

ここでの最後のタスクは簡単です。display_formサブルーチンを使用して、ユーザがクエリーを入力することができるフォームを作成します。このプログラムはダイナミックにフォームを作成します。なぜなら、いくつかの情報（例えば、サーバリスト）は変更されることを前提にしているからです。

```
sub display_form
{
    local ($archie);

    print <<End_of_Archie_One;
Content-type: text/html

<HTML>
<HEAD><TITLE>Gateway to Internet Information Servers</TITLE></HEAD>
<BODY>
<H1>CGI Archie Gateway</H1>
<HR>
<FORM ACTION="/cgi-bin/archie.pl" METHOD="POST">
Please enter a string to search from: <BR>
<INPUT TYPE="text" NAME="query" SIZE=40>
<P>
What archie server would you like to use (<B>please</B>, be considerate
and use the one that is closest to you): <BR>
<SELECT NAME="server" SIZE=1>

End_of_Archie_One

    foreach $archie (sort keys %servers) {
        if ($servers{$archie} eq $default_server) {
            print "<OPTION SELECTED>", $archie, "\n";
```

```
        } else {
            print "<OPTION>", $archie, "\n";
        }
    }
```

このループは連想配列の中を繰り返して、すべてのサーバ名を表示します。

```
    print <<End_of_Archie_Two;
</SELECT>
<P>
Please select a type of search to perform: <BR>
<INPUT TYPE="radio" NAME="type" VALUE="exact" CHECKED>Exact<BR>
<INPUT TYPE="radio" NAME="type" VALUE="ci_sub">Case Insensitive
Substring<BR>
<INPUT TYPE="radio" NAME="type" VALUE="cs_sub">Case Sensitive
Substring<BR>
<P>
<INPUT TYPE="submit" VALUE="Start Archie Search!">
<INPUT TYPE="reset"  VALUE="Clear the form">
</FORM>
<HR>
</BODY>
</HTML>

End_of_Archie_Two
}
```

このダイナミックフォームは図10-2のようになります。

ここではArchieサーバと直接やりとりしていないので、これはむしろ簡単なプログラムです。それでは、もう少し複雑な例を見ていきましょう。

10.7 Web上のネットワークニュース

NNTP（Network News Transfer Protocol）とは、インターネット上でUsenetニュースを転送するときに使用される最も人気のあるソフトウェアです。これは、受信（クライアント）システムが送信（サーバ）システムに対して、どのnewsgroupsを送ってもらうかを伝えたり、各グループからの記事を伝えたりするものです。NNTPは非常に簡単なフォーマットでコマンドを受け付けます。これは投函された記事や臨時のステータス情報から構成されているテキストストリームを送り返します。

CGIゲートウェイはソケットI/Oを使用して、NTTPサーバと直接通信します。このプログラムはnewsgroupsや記事のリストを、その中から選択してユーザに表示します。指定したnewsgroupsから（各記事に対するすべての反響が一緒にグループ分けされている）スレッド形式でニュースを

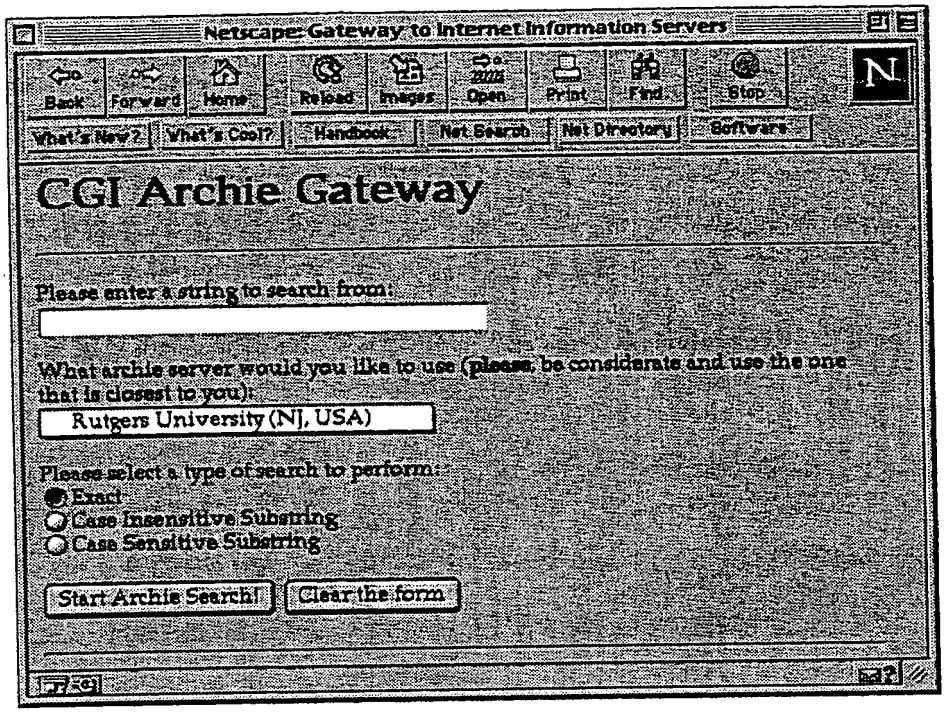


図10-2 Archieフォーム

読むことができます。

```
#!/usr/local/bin/perl

require "sockets.pl";

$webmaster = "Shishir Gundavaram (shishir\@bu\.edu)";
$error = "CGI NNTP Gateway Error";

%groups = ( 'cgi',     'comp.infosystems.www.authoring.cgi',
            'html',    'comp.infosystems.www.authoring.html',
            'images',  'comp.infosystems.www.authoring.images',
            'misc',    'comp.infosystems.www.authoring.misc',
            'perl',    'comp.lang.perl.misc' );
```

groups連想配列には、フォームがダイナミックに作成されたときに表示されるnewsgroupsリストが入っています。

```
$all_groups = '(cgi|html|images|misc|perl)';
```

all_groups変数には、groups連想配列のすべてのキーをリストアップする正規表現が入っています。これは、ユーザが有効なnewsgroupを指定したことを確認するために使用されます。

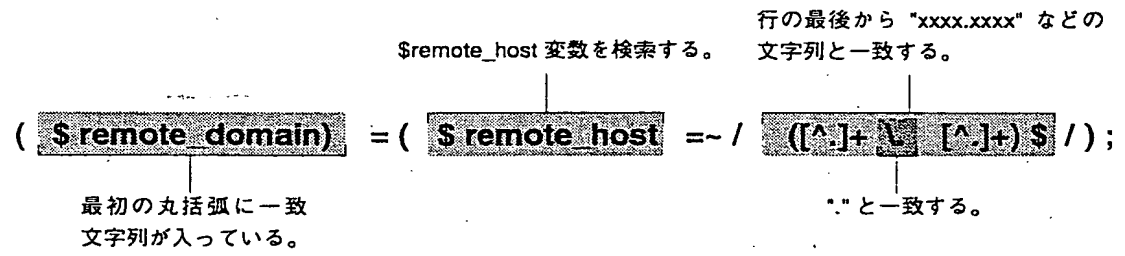
```
$nntp_server = "nntp.bu.edu";
```

NNTPサーバは"nntp.bu.edu"に設定されています。"bu.edu"以外のドメインからこのフォームにアクセスさせたくなければ、以下のような簡単な認証技法を設定することができます。

```
$allowed_domain = "bu.edu";
$remote_host = $ENV{'REMOTE_HOST'};
($remote_domain) = ($remote_host =~ /([^.]+\.[^.]+)$/);

if ($remote_domain ne $allowed_domain) {
    &return_error (500, $error, "Sorry! You are not allowed to read
news!");
}
```

上記で使用されている正規表現は、ホスト名またはIPアドレスからドメイン名を抽出します。



$remote_host 変数に "web.plus.cgi.com" という文字列が入っていると、
$remote_domain 変数には "cgi.com" が入ります。

もしくは、次のように複数のドメインを可能にすることができます。

```
$allowed_domains = "(bu.edu|mit.edu|perl.com)";
$remote_host = $ENV{'REMOTE_HOST'};

if ($remote_host !~ /$allowed_domains$/o) {
    &return_error (500, $error, "Sorry! You are not allowed to read news!");
}
```

プログラムを続けましょう。

```
&parse_form_data (*NEWS);
$group_name = $NEWS{'group'};
$article_number = $NEWS{'article'};
```

このCGIゲートウェイに対するフォームフロントエンドはありません。その代わり、すべてのパラメータがクエリー情報（GETメソッド）として渡されます。クエリーなしでこのアプリケーションにアクセスすると、すべてのnewsgroupsがリストアップされているドキュメントが返されます。このリストからnewsgroupを選択すると、今度は必要なnewsgroupを指定するクエリーを使用して、プログラムが再度呼び出されます。例えば、キーが"images"のnewsgroupが必要なら、次のクエリーがプログラムに渡されます。

```
http://some.machine/cgi-bin/nntp.pl?group=images
```

groups連想配列は、実際のnewsgroup名と文字列"images"とを関連付けます。前述した例では、実際のIPアドレスの代わりにArchieサーバ名が渡されましたが、これはハンドリングのためのより安全な方法です。プログラムが上記のようなクエリーを受け取ると、プログラムはnewsgroupの記事リストを表示します。ユーザが記事を選択すると、クエリー情報は次のようになります。

```
http://some.machine/cgi-bin/nntp.pl?group=images&article=18721
```

このプログラムはこの記事を表示します。

```
if ($group_name =~ /\b$all_groups\b/o) {
   $selected_group = $groups{$group_name};
```

groupフィールドがall_groupsに格納されているような有効なnewsgroup名から構成されていると、このコードブロックが実行されます。実際のnewsgroup名はselected_group変数に格納されます。

```
    &open_connection (NNTP, $nntp_server, "nntp") ||
          &return_error (500, $error, "Could not connect to NNTP server.");

    &check_nntp ();
```

NNTPサーバに対してソケットがオープンされます。通常、このサーバはポート119で実行しています。check_nntpサブルーチンは、サーバが出力したヘッダ情報をチェックします。サーバがエラーメッセージを発行すると、このスクリプトは終了します。

```
    ($first, $last) = &set_newsgroup ($selected_group);
```

NNTPサーバは、任意の数からスタートして昇順で番号を付けていくことにより、newsgroupのすべての記事を記録し続けます。set_newsgroupサブルーチンは先頭記事と最終記事の識別番号を返します。

```
if ($article_number) {
    if (($article_number < $first) || ($article_number > $last)) {
        &return_error (500, $error,
             "The article number you specified is not valid.");
    } else {
        &show_article ($selected_group, $article_number);
    }
```

newsgroupが選択されたときにダイナミックに生成されたリストから、ユーザが記事を選択すると、このコードが実行されます。記事番号がチェックされ、それが有効範囲内に入っているかどうかを確認します。ユーザに表示されるリストはset_newsgroupサブルーチンが生成した範囲をベースにしているのに、なぜここで、これをチェックする必要があるのか疑問に思うかもしれません。その理由は、NNTPサーバは定期的に記事を出力していて、その著者が記事を消去している場合があるからです。リストが表示された時間とユーザが選択した時間の間にかなりのずれがあると、指定した記事番号が無効になっている可能性が出てきます。さらに、ユーザがクエリーをハードコード化した場合もハンドルできるようにしたくなります。

```
} else {
    &show_all_articles ($group_name, $selected_group, $first, $last);
}
```

ユーザがメインHTMLドキュメントからnewsgroupを選択したときに記事が指定されないと、show_all_articlesサブルーチンが呼び出されて、選択したnewsgroupのすべての記事リストを表示します。

```
print NNTP "quit", "\n";
&close_connection (NNTP);
```

最後に、NNTPサーバにquitコマンドが送られて、ソケットがクローズされます。

```
} else {
    &display_newsgroups ();
}

exit (0);
```

このプログラムがクエリー情報なしでアクセスされたり、もしくは指定したnewsgroupがgroups連想配列に格納されているリストの中に入っていないと、display_newsgroupsサブルーチンが呼び出されて有効なnewsgroupsを出力します。

以下のprint_headerサブルーチンは、MIMEヘッダと、タイトルとヘッダを表示するためのHTMLを表示します。

```
sub print_header
{
    local ($title) = @_;

    print "Content-type: text/html", "\n\n";
    print "<HTML>", "\n";
    print "<HEAD><TITLE>", $title, "</TITLE></HEAD>", "\n";
    print "<BODY>", "\n";
    print "<H1>", $title, "</H1>", "\n";
    print "<HR>", "<BR>", "\n";
}
```

print_footerサブルーチンは、webmasterのアドレスを出力します。

```
sub print_footer
{
    print "<HR>", "\n";
    print "<ADDRESS>", $webmaster, "</ADDRESS>", "\n";
    print "</BODY></HTML>", "\n";
}
```

escapeサブルーチンは、英数字キャラクタと空白以外のすべてのキャラクタをエスケープします。これは、特別キャラクタが適切に表示されるようにするためです。

```
sub escape
{
    local ($string) = @_;

    $string =~ s/([^\w\s])/sprintf ("&#%d;", ord ($1))/ge;

    return ($string);
}
```

例えば、newsgroupの記事に次の内容が入っていると、

```
From: joe@test.net (Joe Test)
Subject: I can't get the <H1> headers to display correctly
```

ブラウザは実際に"<H1>"を解釈してしまい、残りのドキュメントがめちゃくちゃになります。このサブルーチンはこのテキストをエスケープして、次のように表示します。

```
From&#58; joe&#64;test&#46;net &#40;Joe Test&#41;
Subject&#58; I can&#39;t get the &#60;H1&#62; headers to display
correctly
```

Webクライアントは&#nという形式の任意の文字列を解釈することができます。ここで、nはキャラクタのASCIIコードです。これは表示が少し遅くなりますが、指定キャラクタだけをエスケープ

するより、はるかに安全です。

check_nntpサブルーチンは、返されるステータスが成功（200または201）もしくは失敗（4xxまたは5xx）のいずれかになるまで、NNTPサーバから出力を連続して読み込みます。これらのステータスコードはHTTPステータスコードによく似ていることに気付いているかもしれません。事実、標準規格にしたがっているほとんどのインターネットサーバでは、このコードを使用しています。

```
sub check_nntp
{
    while (<NNTP>) {
        if (/^(200|201)/) {
            last;
        } elsif (/^4|5\d+/) {
            &return_error (500, $error, "The NNTP server retured an error.");
        }
    }
}
```

set_newsgroupサブルーチンはnewsgroupの先頭記事番号と最終記事番号を返します。

```
sub set_newsgroup
{
    local ($group) = @_;
    local ($group_info, $status, $first_post, $last_post);

    print NNTP "group ", $group, "\n";
```

groupコマンドがNNTPサーバに送られます。これに応答して、サーバは現在のnewsgroupを指定したものに設定し、以下のフォーマットで情報を出力します。

```
group comp.infosystems.www.authoring.cgi
211 1289 4776 14059 comp.infosystems.www.authoring.cgi
```

先頭カラムはこのオペレーションのステータス（成功を表している211）を表しています。記事の総数、先頭記事と最終記事、newsgroup名が後に続いています。見ればわかるように、記事数は先頭記事と最終記事の数の差にはなっていません。これは、（前述したように）記事の更新と消去のためです。

```
    $group_info = <NNTP>;
    ($status, $first_post, $last_post) = (split (/\s+/, $group_info))[0, 2, 3];
```

サーバ出力は空白で分割されています。先頭、3番め、4番めの要素はそれぞれstatus、first_post、last_postに格納されます。配列はゼロベースなので、先頭要素は1ではなくゼロであることを覚えておきましょう。

```
    if ($status != 211) {
        &return_error (500, $error,
                       "Could not get group information for $group.");
    } else {
        return ($first_post, $last_post);
    }
}
```

ステータスが211でなければ、エラーメッセージが表示されます。そうでなければ、先頭記事番号と最終記事番号が返されます。

show_articleサブルーチンでは、実際の新しい記事が検索されプリントされます。

```
sub show_article
{
    local ($group, $number) = @_;
    local ($useful_headers, $header_line);

    $useful_headers = '(From:|Subject:|Date:|Organization:)';

    print NNTP "head $number", "\n";
    $header_line = <NNTP>;
```

headコマンドは指定した記事のヘッダを表示します。以下は、NNTP出力のフォーマットです。

```
221 14059 <47hh6767ghe1$d09@nntp.test.net> head
Path: news.bu.edu!decwrl!nntp.test.net!usenet
From: joe@test.net (Joe Test)
Newsgroups: comp.infosystems.www.authoring.cgi
Subject: I can't get the <H1> headers to display correctly
Date: Thu, 05 Oct 1995 05:19:03 GMT
Organization: Joe's Test Net
Lines: 17
Message-ID: <47hh6767ghe1$d09@nntp.crl.com>
Reply-To: joe@test.net
NNTP-Posting-Host: my.news.test.net
X-Newsreader: Joe Windows Reader v1.28
.
```

先頭行にはステータス、記事番号、メッセージID、NNTPコマンドがそれぞれ入っています。221のステータスは成功を表しています。他の行はすべて、さまざまな記事ヘッダから構成されており、これは記事が投函された方法や場所をベースに構成されています。ヘッダ本体は"."キャラクタで終っています。

```
    if ($header_line =~ /^221/) {
        &print_header ($group);
        print "<PRE>", "\n";
```

サーバが221の成功ステータスを返すと、print_headerサブルーチンが呼び出されて、MIMEヘッダと、その後に通常のHTMLが表示されます。

```
while (<NNTP>) {
    if (/^$useful_headers/) {
        $_ = &escape ($_);
        print "<B>", $_, "</B>";
    } elsif (/^\.\s*$/) {
        last;
    }
}
```

このループはヘッダ本体の中を繰り返し、From、Subject、Date、Organizationのヘッダをエスケープして表示します。

```
print "\n";
print NNTP "body $number", "\n";
<NNTP>;
```

この時点まですべてうまくいくと、bodyコマンドがサーバに送られます。すると、サーバは以下のフォーマットで記事本体を出力します。

```
body 14059
222 14059 <47hh6767ghe1$d09@nntp.test.net> body
I am trying to display headers using the <H> tag, but it does not
seem to be working. What should I do? Please help.

Thanks in advance,
-Joe
.
```

headコマンドがうまく実行されれば、このコマンドのステータスをチェックする必要はありません。サーバは222のステータスを返して成功を表します。

```
while (<NNTP>) {
    last if (/^\.\s*$/);
    $_ = &escape ($_);
    print;
}
```

whileループは本体の中を繰り返し、すべての行をエスケープして、それを表示します。行がピリオドで始まっていて、他には何も入っていないと、ループは終了します。

```
    print "</PRE>", "\n";
    &print_footer ();
} else {
```

```
        &return_error (500, $error,
            "Article number $number could not be retrieved.");
    }
}
```

指定した記事が見つからないと、エラーメッセージが表示されます。

以下のサブルーチンは、メモリの中へ特定グループのすべての記事を読み込んで、それをスレッド（指定した記事に対するすべての返答を、読みやすいようにグループ分けすること）して、記事番号とサブジェクト行を表示します。

```
sub show_all_articles
{
    local ($id, $group, $first_article, $last_article) = @_;
    local ($this_script, %all, $count, @numbers, $article,
           $subject, @threads, $query);

    $this_script = $ENV{'SCRIPT_NAME'};
    $count = 0;
```

これはプログラムの中で一番複雑な（しかし一番おもしろい）部分です。目の前には、NNTPサーバからのまったく基本的な出力からはるかに成長した、すばらしいWebインタフェースが表示されます。

```
    print NNTP "xhdr subject $first_article-$last_article", "\n";
    <NNTP>;
```

xhdr subjectは、以下のフォーマットで指定した範囲のすべての記事をリストアップします。

```
xhdr subject 4776-14059
221 subject fields follow
4776 Re: CGI Scripts (guestbook ie)
4831 Re: Access counter for CERN server
12769 Re: Problems using sendmail from Perl script
12770 File upload, Frames and BSCW
-
- (More Articles)
-
.
```

この先頭行にはステータスが入っています。newsgroupが存在していることがわかっているので、これをチェックする必要はありません。各記事はその番号と表題と一緒にリストアップされます。

```
    &print_header ("Newsgroup: $group");
    print "<UL>", "\n";

    while (<NNTP>) {
```

```
        last if (/^\.\s*$/);
        $_ = &escape ($_);

        ($article, $subject) = split (/\s+/, $_, 2);

        $subject =~ s/^\s*(.*)\s*/$1/;
        $subject =~ s/^[Rr][Ee]:\s*//;
```

このループはすべてのサブジェクトの中を繰り返します。splitコマンドは各エントリーを記事番号とサブジェクトに分割します。行の冒頭にある"Re:"だけでなく、前後のスペースもサブジェクトから削除されます。これはソート目的のために実行されます。

```
        if (defined ($all{$subject})) {
            $all{$subject} = join ("-", $all{$subject}, $article);
        } else {
            $count++;
            $all{$subject} = join ("\0", $count, $article);
        }
    }
```

これは記事をスレッドします。新しい記事は、表題自体によってキーされている連想配列$allに格納されます。$count変数は、配列の各値をスタートするための固有番号を指定します。記事がすでに存在していると、その記事番号は、同じ表題を使用してその要素の最後に追加されます。例えば、表題が次のようになっていると、

```
2020 What is CGI?
2026 How do you create counters?
2027 Please help with file locking!!!
2029 Re: What is CGI?
2030 Re: What is CGI?
2047 Re: How do you create counters?
.
.
.
```

連想配列はこのようになります。

```
$all{'What is CGI?'} = "1\02020-2029-2030";
$all{'How do you create counters?'} = "2\02026-2047";
$all{'Please help with file locking!!!'} = "3\02027";
```

先頭スレッド（"What's CGI?"）には$countに1を割り当てて、2番めのスレッドには2を割り当てて、順次このように割り当てていくことに注意してください。この番号を使用して後でソートして、newsgroupが到着した順番にユーザがスレッドを見れるようにします。

```
    @numbers = sort by_article_number keys (%all);
```

ここでは、一般的なPerlテクニックを使用してソートしています。sortコマンドはサブルーチンを繰り返し呼び出します（この場合、by_article_numberというサブルーチンを記述しました）。高速アルゴリズムを使用して、$all配列から要素ペアをサブルーチンに渡します。

```
    foreach $subject (@numbers) {
        $article = (split("\0", $all{$subject}))[1];
```

このループはすべての表題の中を繰り返します。各表題の記事番号リストがarticleに格納されます。したがって、"What is CGI?"の$article変数は次のようになります。

```
2020-2029-2030
```

それでは、記事の文字列について見ていきましょう。

```
        @threads = split (/-/, $article);
```

特定の表題に関するすべての記事が入っている文字列は、"-"デリミタで分割され、threads配列に格納されます。

```
        foreach (@threads) {
            $query = join ("", $this_script, "?", "group=", $id,
                       "&", "article=", $_);

           print qq|<LI><A HREF="$query">$subject</A>|, "\n";
        }
    }

    print "</UL>", "\n";
    &print_footer ();
}
```

このループは各記事番号（もしくはスレッド）の中を繰り返し、newsgroup名と記事番号が入っているハイパーテキストリンクを作成します（図10-3参照）。

以下は連想配列の2つの値を比較するための簡単なサブルーチンです。

```
sub by_article_number
{
    $all{$a} <=> $all{$b};
}
```

このステートメントは以下と同等です。

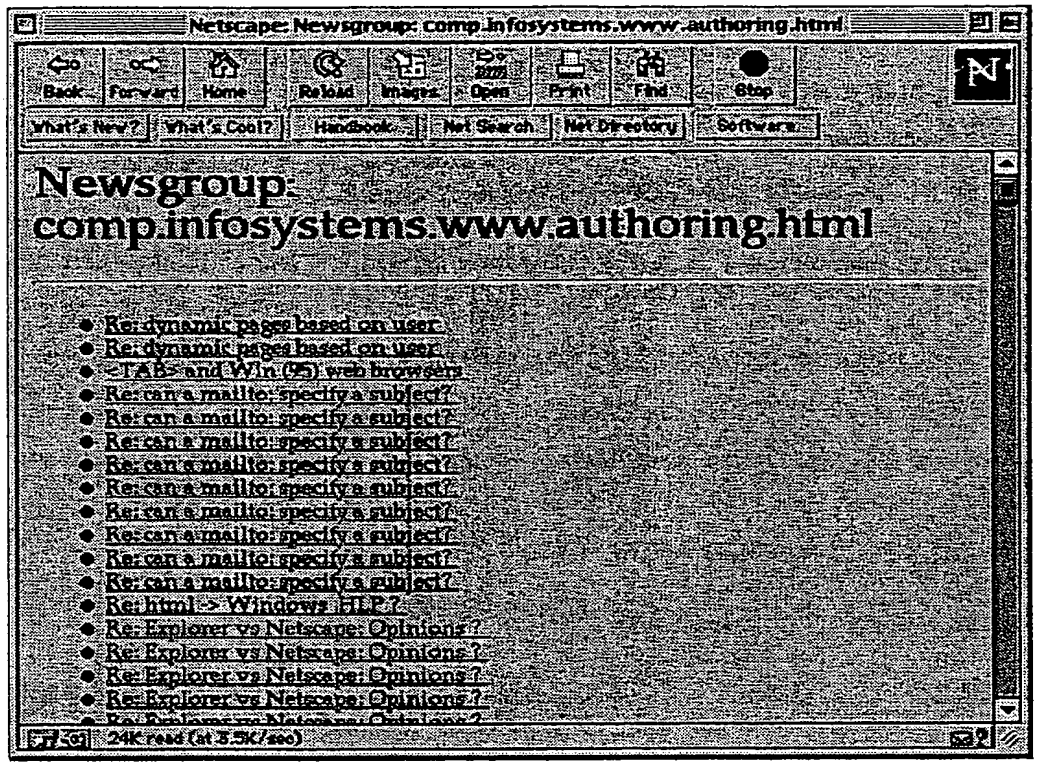


図10–3　ニュース記事

```
if ($all{$a} < $all{$b}) {
    return (-1);
} elsif ($all{$a} == $all{$b}) {
    return (0);
} elsif ($all{$a} > $all{$b}) {
    return (1);
}
```

$aと$bは、連想配列の2つの値を指定します。この場合、Perlはこのロジックを使用して、連想配列のすべての値を比較していきます。

display_newsgroupsサブルーチンは、groups連想配列に入っているすべてのnewsgroupsをリストアップするダイナミックHTMLドキュメントを作成します。

```
sub display_newsgroups
{
    local ($script_name, $keyword, $newsgroup, $query);

    &print_header ("CGI NNTP Gateway");
    $script_name = $ENV{'SCRIPT_NAME'};
```

```
    print "<UL>", "\n";

    foreach $keyword (keys %groups) {
        $newsgroup = $groups{$keyword};
        $query = join ("", $script_name, "?", "group=", $keyword);

        print qq|<LI><A HREF="$query">$newsgroup</A>|, "\n";
    }

    print "</UL>";
    &print_footer ();
}
```

連想配列の指定キーを構成しているクエリーを使用して、各newsgroupが順不同でリストアップされます。2重引用符の代わりに"|"がデリミタになっていることを除けば、qq|...|表記は"..."表記と同じです。

10.8 マジッククッキー（Magic Cookies）

「8章　複数フォームとのやりとり」では、複数のフォームを扱う場合の問題点について説明し、いくつかの解決策を紹介しました。本章では、クライアントとサーバに関する新たに学んだ知識を駆使して、再度この問題を取り上げていきます。

複数フォームから構成されているインタフェースは、CGIに関する難しい問題を含んでいます。異なるフォーム上に格納された情報を、どのようにして記憶しておけばよいのでしょうか。（ローカルマシン上で稼働している）通常のグラフィカルインタフェースアプリケーションでは、図10-4に示すように、ただフォームを表示してその結果を格納しているだけです。

クライアントとサーバが別れていないときに、連続したフォームから情報を格納することは簡単です。しかし、CGIを使用する場合、フォームが発行されるたびに、サーバは繰り返しプログラムを呼び出します。1つの実行中のプログラムがあるのではなく、図10-5に示したように、複数のインスタンスがあるのです。

直面する問題は、前の実行で検索されたデータを、プログラムの各インスタンスにどのようにして伝えていけばよいかという点です。

テンポラリファイルは簡単な解決策ですが、これは面倒です。プログラムは読み書きするたびにファイルを把握しておく必要があります。複数ユーザが同時にプログラムを実行している場合には、正しいファイルを把握しておくことはかなり大変です。その上、ファイルがシステム上で表示可能になっているので、情報は安全であるとは言えません。ファイルにアクセスするには時間がかかるので、処理も遅くなります。最後に、ユーザがセッションを終了せずに立ち去った場合には、ファ

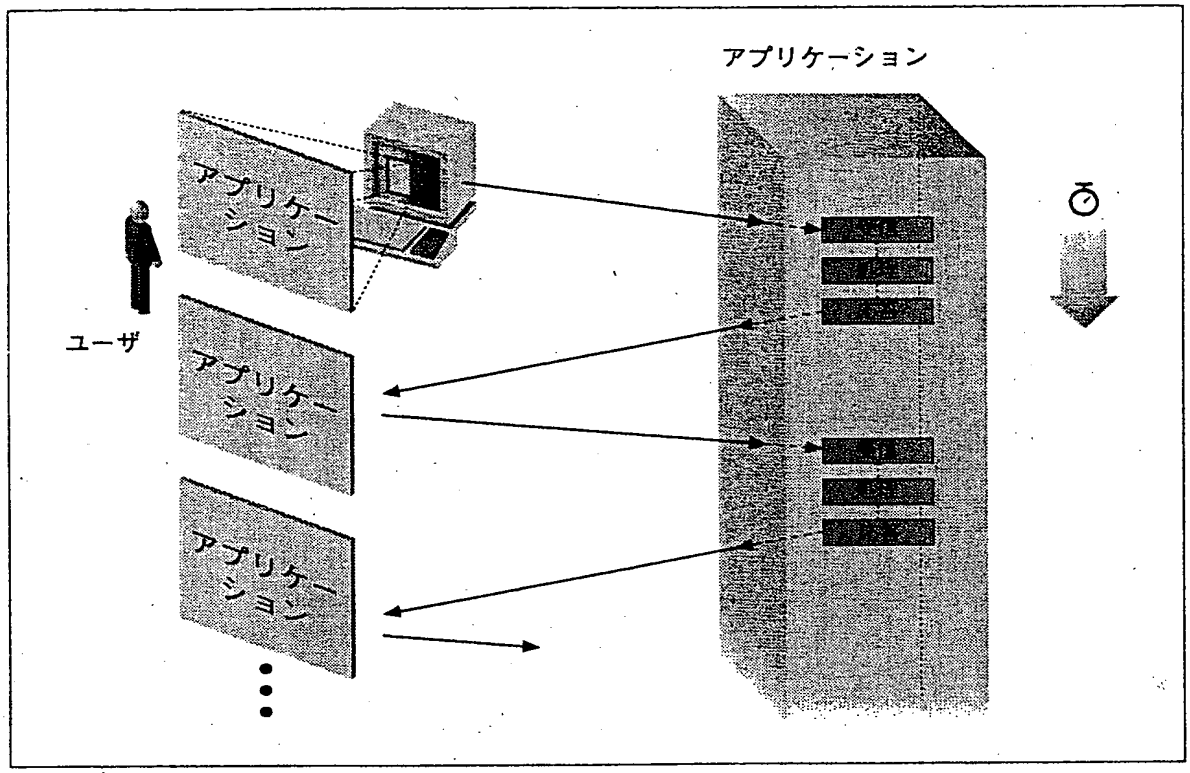


図10-4　複数フォームを扱っているローカルアプリケーション

イルを一掃することを覚えておかなければなりません。

よりすぐれた解決策では特別なサーバを使用します。このサーバはCGIプログラムの状態を維持管理し、他のサーバ同様、連続して動作します。すべてのタイプと目的を持ったCGIプログラムがこのサーバを使用して、情報を格納します。このサーバがテンポラリファイルよりも優れている点は、データはメモリ上にあるという点です。これにより、操作は速くなり、データをより安全に維持することができます。

このサーバアプローチの核心は、1つのプログラムのインスタンスがサーバに送ったデータを取り出す方法を、CGIプログラムが把握しているという点にあります。プログラムの各インスタンスは、データを見つけ出すためのハンドルが必要です。このアクセスを行うために、サーバは、CGIプログラムを実行する各ユーザに独自の識別子を関連付けます。データを格納するときに、プログラムはこの識別子を適用し、データを取り出すときに、プログラム別のインスタンスに再度その識別子を適用します。派手な言語を好む、コンピュータの人々はそのような識別子をマジッククッキー（“magic cookies”）と呼びます。1つのクッキーを使用すれば、CGIプログラムは任意のデータ量を維持管理することができます。したがって、サーバはクッキーサーバと呼ばれ、CGIプログラム

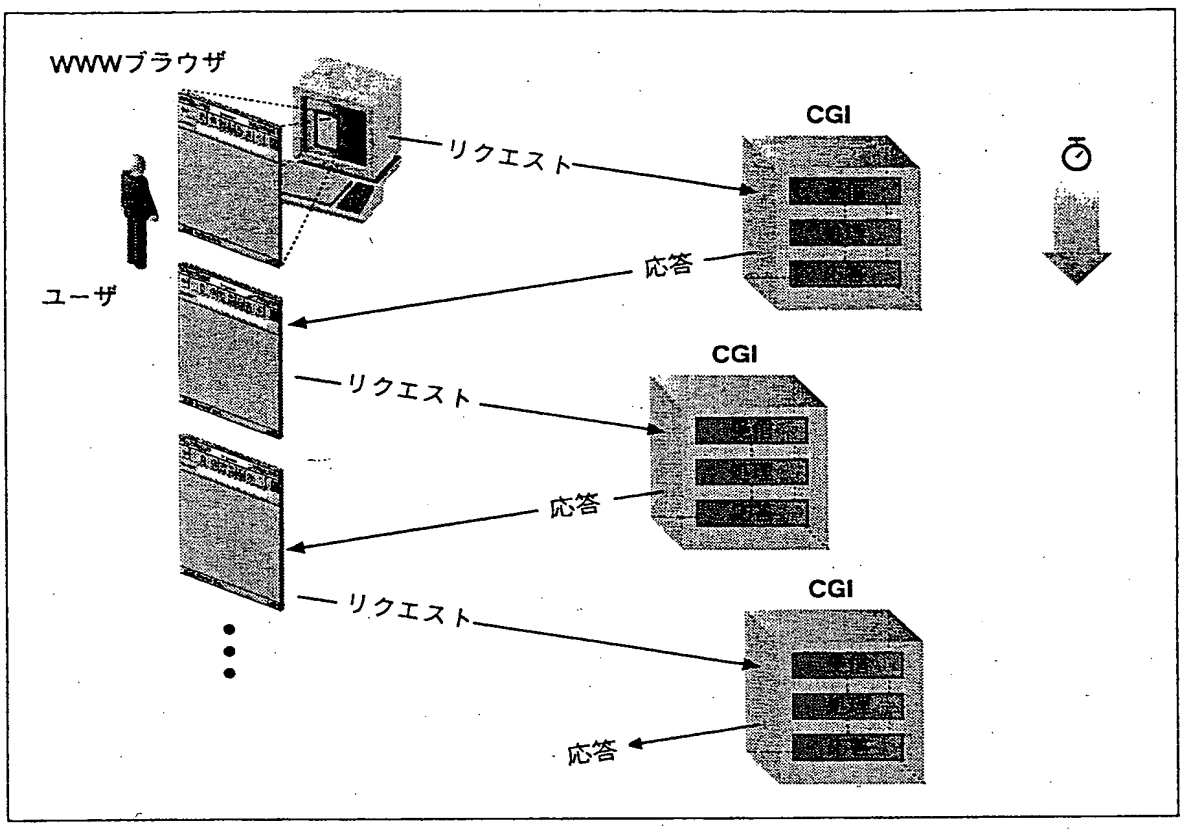


図10-5 サーバ上の複数フォーム

はクッキークライアントと呼ばれています。

もう1つの大きな問題も、クッキーを使用して解決する必要があります。CGIプログラムの1つのインスタンスは次のインスタンスへクッキーを渡す必要があります。図10-5を見ると、矢印でソリューションがわかるかもしれません。クッキーを次のフォームに渡し、それをフォームに戻します。これが本書で使用した解決策です。CGIプログラムが各フォームを作成する場合、隠ぺいフィールドにクッキーを埋め込みます。ユーザがそのフォームを発行すると、隠ぺいフィールドを戻します。これがスタートすると、プログラムの新しいインスタンスは任意の他のフィールドと同様にクッキーを検索し、サーバにデータを問い合わせます。このプロシージャは図10-6のようになっています。

それではクッキーと関係するデータの流れについて、ひととおりのセッションを通じて見ていきましょう。

- ユーザが先頭フォームに記入して、CGIプログラムが初めて呼び出されます。
- CGIプログラムは初めてサーバと連絡を取ります。サーバはクッキーを作成し、それをプログラムに渡します。サーバが与えたクッキーを使用して、プログラムはデータをサーバに渡します。

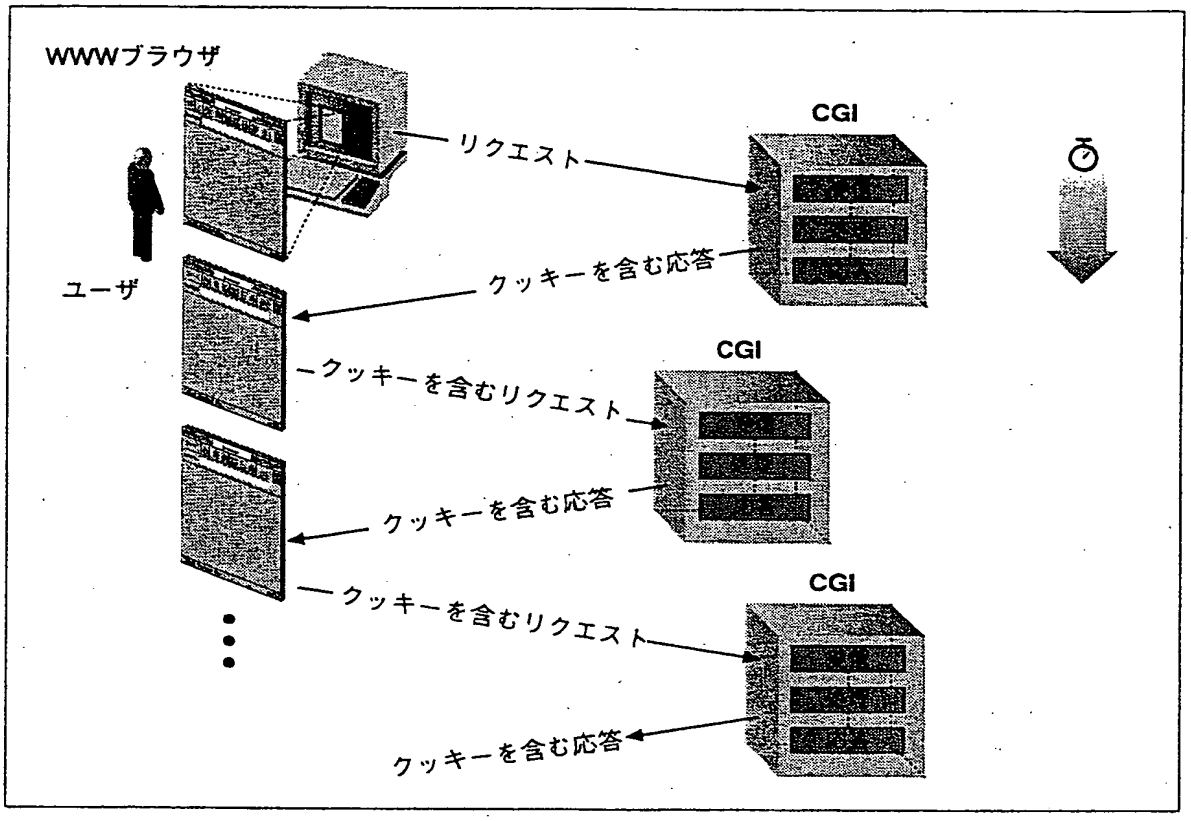


図10-6　Webクライアントとサーバにおけるクッキーサーバのやりとり

- プログラムはユーザ向けの次のフォームを作成し、隠ぺいフィールドにクッキーを埋め込んで、そのフォームをブラウザに送ります。
- ブラウザはフォームを表示します。そのフォームにユーザが記入して発行します。フォームはクッキーを使用して隠ぺいフィールドを戻します。
- CGIプログラムの新しいインスタンスが開始します。これはフォームデータからクッキーを獲得し、再度、サーバと連絡を取ります。今回は、プログラムは新しいクッキーを作成する代わりに、既存クッキーを渡します。

これがここでの戦略です。これを理解すれば、この後のコードを理解できるようになるはずです。

10.9　サーバを使用して状態を維持管理する

「8章　複数フォームとのやりとり」では、複数フォーム間で情報を維持管理するためのテクニックについて、いくつか取り上げました。そこでは、テンポラリファイル、隠ぺい変数、Netscape

Persistent Cookiesを使用しています。ここでは、状態を維持管理するための他のメソッドについて見ていきましょう。この場合、情報を格納して検索するには、クッキーサーバと通信する必要があります。

クッキーを使用する実際のプログラムを見れば、クッキーがどのように動作するのか理解しやすくなるでしょう。そこで、2つのフォームを表示し、クッキーサーバを呼び出し、そこで返ってきた情報を格納するためのCGIプログラムについて見ていきましょう。以下に先頭フォームを示します。

```
<HTML>
<HEAD><TITLE>College/School Survey</TITLE></HEAD>
<BODY>
<H1>Interests</H1>
<HR>
<FORM ACTION="/cgi-bin/cookie_client.pl?next=/location.html" METHOD="POST">
```

ACTION属性は、クエリー文字列として一連の次のフォームを指定します。ファイル名はドキュメントルートディレクトリに相対的になっています。

```
<INPUT TYPE="hidden" NAME="Magic_Cookie" VALUE="-*Cookie*-">
```

文字列"-*Cookie*-"は、このフォームがCGIプログラムで解析されたときに、ランダムクッキー識別子で置換されます。このクッキーはフォーム情報を識別するときに使用されます。

```
What subject are you interested in? <BR>
<INPUT TYPE="text" NAME="subject" SIZE=40>
<P>
What extra-curricular activity do you enjoy the most? <BR>
<INPUT TYPE="text" NAME="interest" SIZE=40>
<P>
<INPUT TYPE="submit" VALUE="See Next Form!">
<INPUT TYPE="reset"  VALUE="Clear the form">
</FORM>
<HR>
</BODY>
</HTML>
```

以下はこのシリーズの2番めのフォームです。これはlocation.htmlという名前のファイルで格納されます。なぜなら、先頭フォームのACTION属性に、この名前が指定されているからです。

```
<HTML>
<HEAD><TITLE>College/School Survey</TITLE></HEAD>
<BODY>
<H1>Location</H1>
<HR>
<FORM ACTION="/cgi-bin/cookie_client.pl" METHOD="POST">
```

これはこのシリーズの最後のフォームなので、プログラムにクエリー情報は渡されません。

```
<INPUT TYPE="hidden" NAME="Magic_Cookie" VALUE="-*Cookie*-">
Where would you like to go to school? <BR>
<INPUT TYPE="text" NAME="city" SIZE=40>
<P>
What type of college do you prefer? <BR>
<INPUT TYPE="text" NAME="type" SIZE=40>
<P>
<INPUT TYPE="submit" VALUE="Get Summary!">
<INPUT TYPE="reset"  VALUE="Clear the form">
</FORM>
<HR>
</BODY>
</HTML>
```

これらのプログラムを正しくハンドルするためのプログラムを今すぐ見ていくのはやめて、この事例では少し変わったことを実行してみましょう。ここでは、クッキーサーバについて見ていきます。このサーバは、CGIプログラムの状態を維持管理するための連続的に動作するプログラムです。その後で、フォームを解析するプログラム（クッキークライアント）に戻って、サーバとのやりとりについて説明します。

10.9.1　クッキーサーバ

ここでは、ローカルマシン上で稼働するCGIプログラムの汎用サーバを紹介します。各CGIプログラムがクッキークライアントです。これを接続すると、このサーバはコマンドを受け付けるための長いループに入ります。これはCGIスクリプトではないことに注意してください。その代わり、これはCGIスクリプトのためのデータストレージサービスを提供します。

```
#!/usr/local/bin/perl

require "sockets.pl";
srand (time|$$);
```

srand関数は、乱数シードを設定します。現在時刻とプロセス識別番号（PID）の論理和は非常に優れたシードを作成します。

```
$HTTP_server = "128.197.27.7";
```

CGIスクリプトがこのサーバに接続するHTTPサーバのIPアドレスが指定されます。これを使用して、Web上にある他のHTTPサーバ上で実行しているCGIプログラムがこのサーバと通信しないようにします。

```
$separator = "\034";
$expire_time = 15 * 60;
```

expire_time変数はクッキーが有効である時間を（秒数で）設定します。この場合、クッキーは15分間有効となります。

```
%DATA = ();
$max_cookies = 10;
$no_cookies = 0;
```

DATA連想配列は、フォーム情報を保持するために使用されます。max_cookies変数は、一度にアクティブになれるクッキー数の限度を設定します。no_cookie変数はカウンタで、アクティブクッキー数を記録し続けます。

```
$error = 500;
$success = 200;
```

この2つの変数は、それぞれエラーと成功のステータスコードを保持します。

```
$port = 5000;
&listen_to_port (SOCKET, $port) || die "Cannot create socket.", "\n";
```

listen_to_port関数はソケットライブラリの一部です。これは指定したポート上で可能な接続を待ち受けます。この場合、ポート番号5000が使用されています。ただし、サーバを設定するポートがわからなければ、ソケットライブラリに問い合わせることができます。

```
( ($port) = &listen_to_port (SOCKET) ) || die "Cannot create socket.", "\n";
print "The Cookie Server is running on port number: $port", "\n";
```

listen_to_port関数が（1つの引数を使用して）これと同様に呼び出されると、空のポートが選択されます。そして、現在のポート番号を反映するために、クッキークライアントを修正する必要が出てきます。もしくは、クライアントが“cookie”という名前を使用してサーバを参照することができるようになれば、/etc/servicesファイルにクッキーサーバのエントリーを作成するようにシステム管理者に問い合わせることができます。

```
while (1) {
    ( ($ip_name, $ip_address) = &accept_connection (COOKIE, SOCKET) )
        || die "Could not accept connection.", "\n";
```

これは接続を連続して受け付ける無限ループを開始します。接続が確立されると、新しいソケットハンドルCOOKIEが作成され処理を行います。一方、元のファイルハンドルSOCKETはさらに

接続を受け付けるために戻されます。accept_connectionサブルーチンはリモートホストのアドレスとホスト名を返します。この場合、これは常にHTTPサーバのアドレスを指し示します。なぜなら、CGIプログラム（またはクライアント）はそのサーバから実行されているからです。

このように実装されたクッキーサーバは、一度に1つの接続相手とだけ対話することができます。他のすべての接続は待ち行列の中に入り、受信された順番にハンドルされていきます（複数の接続を同時にハンドルすることができるサーバの実装方法については、後で説明します）。

```
select (COOKIE);
$cookie = undef;
```

デフォルトの出力ファイルハンドルはCOOKIEに設定されます。cookie変数は現在のクッキー識別子を保持するのに使用されます。

```
if ($ip_address ne $HTTP_server) {
    &print_status ($error, "You are not allowed to connect to server.");
```

リモートホストのIPアドレスがHTTPサーバのアドレスと一致しなければ、その接続は他のホストからやってきたものです。他のホスト上で実行しているサーバをこのサーバに接続して情報を格納すると、重大なシステムオーバーロードを引き起こす可能性があるので、そうはしたくありません。ただし、ドメイン内のすべてのマシンがこのサーバにアクセスして情報を格納することができるように、これを設定することは可能です。

```
} else {
    &print_status ($success, "Welcome from $ip_name ($ip_address)");
```

接続が正しい場所（自分たちのHTTPサーバ）からやってきたものであれば、歓迎メッセージが表示されます。print_statusサブルーチンはステータス番号とメッセージを標準出力に出力します。

```
    while (<COOKIE>) {
        s/[\000-\037]//g;
        s/^\s*(.*)\b\s*/$1/;
```

whileループはソケットから連続的に入力を受け付けます。前後のスペースはもちろん、すべてのコントロールキャラクタも入力から削除されます。このサーバは以下のコマンドを受け付けます。

```
new remote-address
cookie cookie-identifier remote-address
key = value
list
delete
```

これについてちょっと説明しましょう。

```
if ( ($remote_address) = /^new\s*(\S+)$/) {
```

newコマンドは新しい固有のクッキーを作成し、それをソケットに出力します。HTTPサーバに接続されているホストのリモートアドレスは、引数としてこのコマンドに渡されます。こうすることで、侵入者がサーバに入り込むのを困難にしています。このコマンドの使用法と典型的な出力の例を以下に示します（クライアントのコマンドは太字で表しています）。

```
new www.test.net
200: 13fGK7KIlZSF2
```

固有のクッキー識別子が入ったステータスが出力されます。クライアントはこの行を解析し、クッキーを取得し、クエリーまたは隠蔽変数のどちらかとしてフォームにこれを挿入します。

```
if ($cookie) {
    &print_status ($error, "You already have a cookie!");
```

クッキー変数が定義されていると、エラーメッセージが表示されます。同じセッションで新しいコマンドを何回も呼び出そうとした場合のみ、これが発生します。

```
} else {
    if ($no_cookies >= $max_cookies) {
        &print_status ($error, "Cookie limit reached.");
    } else {
        do {
            $cookie = &generate_new_cookie ($remote_address);
        } until (!$DATA{$cookie});
```

クッキーがこのセッションで定義されていなければ、そしてクッキー数が事前に定義した制限を越えていなければ、generate_new_cookieサブルーチンが呼び出されて独自のクッキーを作成します。

```
        $no_cookies++;
        $DATA{$cookie} = join("::", $remote_address,
                                    $cookie, time);
        &print_status ($success, $cookie);
    }
}
```

クッキーが無事に作成されると、カウンタがインクリメントされ、新しいキーがDATA連想配列に挿入されます。このキーの値は（後でこれをチェックできるように）リモートアドレス、クッキー、（有効期間のための）時間が入った文字列になっています。

```
} elsif ( ($check_cookie, $remote_address) =
    /^cookie\s*(\S+)\s*(\S+)/) {
```

cookieコマンドはそのセッション用クッキーを設定します。クッキーをいったん設定すると、情報を格納したり、格納した情報をリストアップしたり、クッキーを消去したりすることができます。すでに（newコマンドを使用して）有効なクッキーを持っていると、クッキーコマンドが通常使用されます。これは典型的なクッキーコマンドです。

```
cookie 13fGK7KIlZSF2 www.test.net
200: Cookie 13fGK7KIlZSF2 set.
```

サーバは成功または失敗のいずれかを表しているステータスを返します。存在しないクッキーを設定しようとすると、以下のエラーメッセージが表示されます。

```
cookie 6bseVEbhf74 www.test.net
500: Cookie does not exist.
```

また、IPアドレスが、クッキーを作成したときに使用したものと一致しなければ、それが表示されます。

```
cookie 13fGK7KIlZSF2 www.joe.net
500: Incorrect IP address.
```

プログラムを続けましょう。

```
if ($cookie) {
    &print_status ($error, "You already specified a cookie.");
```

cookieコマンドがセッションで何回も指定されると、エラーメッセージが出力されます。

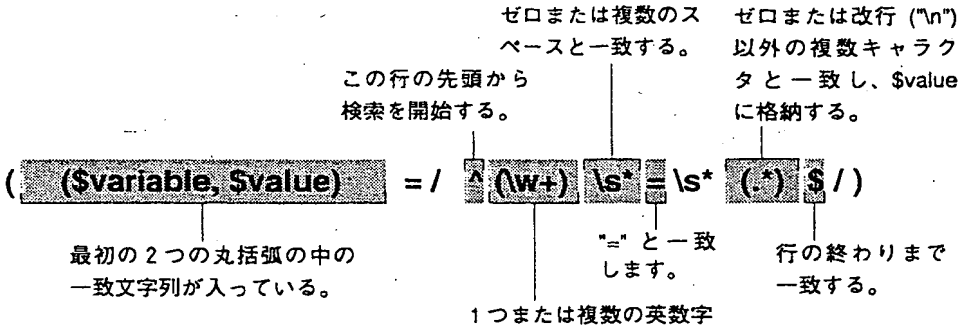
```
} else {
    if ($DATA{$check_cookie}) {
        ($old_address) = split(/::/, $DATA{$check_cookie});

        if ($old_address ne $remote_address) {
            &print_status ($error, "Incorrect IP address.");
        } else {
            $cookie = $check_cookie;
            &print_status ($success, "Cookie $cookie set.");
        }
    } else {
        &print_status ($error, "Cookie does not exist.");
    }
}
```

クッキーが存在していれば、指定したアドレスは元のIPアドレスと比較されます。もしすべて有効なら、cookie変数にはそのクッキーが入ります。

```
} elsif ( ($variable, $value) = /^(\w+)\s*=\s*(.*)$/) {
```

この正規表現は、情報を格納するときに使用されるキーと値が入っているステートメントをチェックします。



この式は、以下のような "assignment" と一致します。

```
name=Larry Bird
team=Boston Celtics!
number=33
```

2つの変数が格納されているサンプルセッションを以下に示します。

```
cookie 13fGK7KIlZSF2 www.test.net
200: Cookie 13fGK7KIlZSF2 set.
name = Joe Test
200: name=Joe Test
organization = Test Net
200: organization=Test Net
```

サーバは厳重になっており、英数字キャラクタ（A–Z、a–z、0–9、_）から構成されている変数だけ受け付けます。

```
if ($cookie) {
    $key = join ($separator, $cookie, $variable);
    $DATA{$key} = $value;
    &print_status ($success, "$variable=$value");
} else {
    &print_status ($error, "You must specify a cookie.");
}
```

変数名は、連想配列のキーを作成するために、クッキーとセパレータと一緒につながっています。

```
} elsif (/^list$/) {
    if ($cookie) {
        foreach $key (keys %DATA) {
            $string = join ("", $cookie, $separator);

            if ( ($variable) = $key =~ /^$string(.*)$/) {
                &print_status ($success, "$variable=$DATA{$key}");
            }
        }
        print ".", "\n";

    } else {
        &print_status ($error, "You don't have a cookie yet.");
    }
```

listコマンドは、DATA連想配列の中を繰り返すことにより、格納された情報すべてを表示します。セパレータが入っているキーだけが出力されます。言い換えると、クッキー、リモートアドレス、時間が入っている最初のキーは表示されません。以下に、listコマンドの出力を示します。

```
cookie 13fGK7KI1ZSF2 www.test.net
200: Cookie 13fGK7KI1ZSF2 set.
list
200: name=Joe Test
200: organization=Test Net
.
```

データは"."キャラクタで終っています。クライアントは、このポイントで読み込みをストップすることができるので、無限ループにはなりません。

```
} elsif (/^delete$/) {
    if ($cookie) {
        &remove_cookie ($cookie);
        &print_status ($success, "Cookie $cookie deleted.");
    } else {
        &print_status ($error, "Select a cookie to delete.");
    }
```

deleteコマンドは内部データベースからクッキーを削除します。remove_cookieサブルーチンが呼び出され、そのクッキーに関連した情報はすべて削除されます。以下は、deleteコマンドの効果を示した例です。

```
cookie 13fGK7KI1ZSF2 www.test.net
200: Cookie 13fGK7KI1ZSF2 set.
list
200: name=Joe Test
200: organization=Test Net
.
delete
```

```
200: Cookie 13fGK7KIlZSF2 deleted.
list
.
```

プログラムを続けましょう。

```
            } elsif (/^exit|quit$/) {
                $cookie = undef;
                &print_status ($success, "Bye.");
                last;
```

exitコマンドとquitコマンドを使用してサーバを終了します。cookie変数はクリアされます。これは非常に重要です。クリアされないと、新しい接続が確立されたときに、サーバは誤って、クッキーはすでに設定されていると判断してしまいます。これは危険なことであり、listコマンドを実行すると、新しいセッションは前の接続で格納した変数を表示してしまいます。

```
            } elsif (!/^\s*$/) {
                &print_status ($error, "Invalid command.");
            }
        }
    }
```

指定したコマンドがリストの中に入っていないと、エラーメッセージが出力されます。

```
    &close_connection (COOKIE);
    &expire_old_cookies();
}

exit(0);
```

サーバとクライアント間の接続がクローズされます。expire_old_cookiesサブルーチンは、期限の過ぎたクッキー（とその関連情報）を削除します。実際には、クッキーは事前に定義した時間を過ぎると必ず削除されるのではなく、接続が終了したときにチェックされ削除されるのです。

print_statusサブルーチンはステータスとメッセージを表示します。

```
sub print_status
{
    local ($status, $message) = @_;

    print $status, ": ", $message, "\n";
}
```

generate_new_cookieサブルーチンは、現在時刻とリモートアドレスをベースにした文字列を暗号化するためのcrypt関数を使用して、ランダムで固有なクッキーを生成します。クッキーの作成

で使用されているアルゴリズムは任意です。ランダムクッキーを生成するには、たいていのアルゴリズムが使用可能です。

```
sub generate_new_cookie
{
    local ($remote) = @_;
    local ($random, $temp_address, $cookie_string, $new_cookie);

    $random = rand (time);
    ($temp_address = $remote) =~ s/\.//g;
    $cookie_string = join ("", $temp_address, time) / $random;
    $new_cookie = crypt ($cookie_string, $random);

    return ($new_cookie);
}
```

expire_old_cookiesサブルーチンは事前に定義した時間が経過したクッキーを削除します。foreachループは、セパレータが入っていないキーを検索しながら、連想配列の中を繰り返します。オリジナルキーごとに、作成時刻と有効時間の合計（秒数）が現在時刻と比較されます。クッキーが期限切れになっていると、remove_keyサブルーチンが呼び出されクッキーを消去します。

```
sub expire_old_cookies
{
    local ($current_time, $key, $cookie_time);

    $current_time = time;

    foreach $key (keys %DATA) {
        if ($key !~ /$separator/) {
            $cookie_time = (split(/::/, $DATA{$key}))[2];

            if ( $current_time >= ($cookie_time + $expire_time) ) {
               &remove_cookie ($key);
            }
        }
    }
}
```

remove_cookieサブルーチンはクッキーを消去します。

```
sub remove_cookie
{
    local ($cookie_key) = @_;
    local ($key, $exact_cookie);

    $exact_cookie = (split(/::/, $DATA{$cookie_key}))[1];

    foreach $key (keys %DATA) {
```

```
        if ($key =~ /$exact_cookie/) {
            delete $DATA{$key};
        }
    }

    $no_cookies--;
}
```

このループは配列の中を繰り返して、クッキー識別子が入っているすべてのキーを検索し、それを消去します。クッキーが削除されると、カウンタがデクリメントされます。

それでは、状態を維持するためにこのサーバと通信するCGIプログラムについて見ていきましょう。

10.9.2 クッキークライアント

それでは、クッキークライアントとは何か、そしてこれはサーバから何を必要とするのかについて見ていきましょう。クライアントとは、各ユーザのために何回も実行する必要があるCGIプログラムのことです（通常、CGIプログラムは複数のフォームを表示し、そのフォームごとに呼び出されます）。このプログラムはクッキーサーバへの接続をオープンし、クッキーを作成し、そこに情報を格納しなければなりません。あるフォームに格納された情報は、ユーザが別のフォームを発行したときに後で取り出されます。

```
#!/usr/local/bin/perl

require "sockets.pl";

$webmaster = "Shishir Gundavaram (shishir\@bu\.edu)";
$remote_address = $ENV{'REMOTE_ADDR'};
```

このHTTPサーバに接続されているホストのリモートアドレスが格納されます。この情報は固有のクッキーを作成するときに使用されます。

```
$cookie_server = "cgi.bu.edu";
$cookie_port = 5000;

$document_root = "/usr/local/bin/httpd_1.4.2/public";
$error = "Cookie Client Error";

&parse_form_data (*FORM);
$start_form = $FORM{'start'};
$next_form = $FORM{'next'};
$cookie = $FORM{'Magic_Cookie'};
```

最初、ブラウザは、先頭フォームを示しているこのプログラムにクエリーを渡す必要があります。

```
http://some.machine/cgi-bin/cookie_client.pl?start=/interests.html
```

その後のすべてのフォームは、<FORM> タグに次のクエリーを入れておく必要があります。

```
<FORM ACTION="/cgi-bin/cookie_client.pl?next=/location.html"
METHOD="POST">
```

ネームクエリーに渡されるファイル名は、フォームごとに異なっていても構いません。フォームがユーザをナビゲートするように指定することができます。

最後に、クッキーが入っているフォームは隠蔽フィールドにしなければなりません。

```
<INPUT TYPE="hidden" NAME="Magic_Cookie" VALUE="-*Cookie*-">
```

このスクリプトは文字列"-*Cookie*-"をクッキーサーバから検索された固有のクッキーに置き換えます。この識別子を使用すると、あるフォームは別のフォームが格納している内容を検索することができるようになります。

このクッキーテクニックは、次のように考えることができます。クッキーサーバは、このプログラムがセーブしたいすべてのデータを格納します。そのデータを検索するには、プログラムの実行ごとにクッキーを把握しておく必要があります。プログラムのインスタンスは、フォームにクッキーを配置することによりクッキーを次のインスタンスに渡します。そして、フォームはそのクッキーをプログラムの新しいインスタンスに送ります。

```
if ($start_form) {
    $cookie = &get_new_cookie ();
    &parse_form ($start_form, $cookie);
```

指定したフォームがシリーズの先頭フォームなら、get_new_cookie サブルーチンが呼び出され、新しいクッキー識別子を検索します。parse_form サブルーチンが隠蔽フィールドに実際のクッキーを配置します。

```
} elsif ($next_form) {
    &save_current_form ($cookie);
    &parse_form ($next_form, $cookie);
```

$start_form もしくは$next_form のいずれかが設定されますが、ブラウザは両方は設定しません。セッションを始められるのは1つだけです。フォームに次のクエリーが入っていると、その中の情報がクッキーサーバに格納されます。これは save_current_form サブルーチンによって実行されます。

```
} else {
    if ($cookie) {
        &last_form ($cookie);
    } else {
        &return_error (500, $error,
```

```
                "You have executed this script in an invalid manner.");
    }
}

exit (0);
```

最後に、フォームにクエリー情報が入っていないのに、クッキー識別子が入っていると、last_formサブルーチンが呼び出され、すべての格納した情報を表示します。

これはメインプログラムの終りの部分です。これは構造体をレイアウトしています。各フォームに正しいスタートもしくは次のクエリーが入っていると、プログラムはユーザが必要なときにすべてを表示します。

open_and_checkサブルーチンはクッキーサーバに接続して、サーバが出力した（末尾の改行キャラクタを削除した）先頭行を読み込みます。そして、この行をチェックして、サーバが適切に機能していることを確認します。

```
sub open_and_check
{
    local ($first_line);

    &open_connection (COOKIE, $cookie_server, $cookie_port)
        || &return_error (500, $error, "Could not connect to cookie server.");

    chop ($first_line = <COOKIE>);

    if ($first_line !~ /^200/) {
        &return_error (500, $error, "Cookie server returned an error.");
    }
}
```

get_new_cookieサブルーチンはサーバにnewコマンドを発行して、そのステータスをチェックして、サーバが固有のクッキー識別子を出力したことを確認します。

```
sub get_new_cookie
{
    local ($cookie_line, $new_cookie);

    &open_and_check ();
    print COOKIE "new ", $remote_address, "\n";
    chop ($cookie_line = <COOKIE>);
    &close_connection (COOKIE);

    if ( ($new_cookie) = $cookie_line =~ /^200: (\S+)$/) {
        return ($new_cookie);
    } else {
        &return_error (500, $error, "New cookie was not created.");
    }
```

```
}
```

parse_formサブルーチンはダイナミックフォームを構築して表示します。これは、例えばlocation.htmlといったファイルからフォームの内容をすべて読み込みます。このサブルーチンが行う唯一の修正は、文字列"-*Cookie*-"をクッキーサーバが返す固有のクッキーに置き換えることです。フォームは入力データとしてクッキーをプログラムに渡し、プログラムはそのクッキーをサーバに渡して、データを設定してリストアップします。

```
sub parse_form
{
    local ($form, $magic_cookie) = @_;
    local ($path_to_form);

    if ($from =~ /\.\./){
       $return_error(500, $error,"what are you trying to go?");
    }
    $path_to_form = join ("/", $document_root, $form);

    open (FILE, "<" . $path_to_form)
        || &return_error (500, $error, "Could not open form.");

    print "Content-type: text/html", "\n\n";

    while (<FILE>) {
        if (/-\*Cookie\*-/) {
            s//$magic_cookie/g;
        }
        print;
    }

    close (FILE);
}
```

save_current_formサブルーチンはクッキーサーバにフォーム情報を格納します。

```
sub save_current_form
{
    local ($magic_cookie) = @_;
    local ($ignore_fields, $cookie_line, $key);

    $ignore_fields = '(start|next|Magic_Cookie)';

    &open_and_check ();
    print COOKIE "cookie $magic_cookie $remote_address", "\n";
    chop ($cookie_line = <COOKIE>);
```

cookieコマンドがサーバに発行され、その後の追加、消去、リストアップの各動作のためのクッキーを設定します。

```
    if ($cookie_line =~ /^200/) {
        foreach $key (keys %FORM) {
            next if ($key =~ /\b$ignore_fields\b/o);

            print COOKIE $key, "=", $FORM{$key}, "\n";
            chop ($cookie_line = <COOKIE>);

            if ($cookie_line !~ /^200/) {
                &return_error (500, $error, "Form info. could not be stored.");
            }
        }
    } else {
        &return_error (500, $error, "The cookie could not be set.");
    }

    &close_connection (COOKIE);
}
```

foreachループは、フォーム情報が入っている連想配列の中を繰り返します。start、next、Magic_Cookieを除くすべてのフィールドがクッキーサーバに格納されます。これら除いたフィールドはこのプログラムによって内部的に使用され、格納されるわけではありません。サーバが情報を格納できないと、エラーを返します。

last_formサブルーチンは、このシリーズの最終フォームが処理されるときに実行されます。listコマンドがサーバに送られます。display_all_itemsサブルーチンは、このコマンドに対応したサーバ出力を読み込んで表示します。最後に、クッキーが消去されます。

```
sub last_form
{
    local ($magic_cookie) = @_;
    local ($cookie_line, $key_value, $key, $value);

    &open_and_check ();
    print COOKIE "cookie $magic_cookie $remote_address", "\n";
    chop ($cookie_line = <COOKIE>);

    if ($cookie_line =~ /^200/) {
        print COOKIE "list", "\n";
        &display_all_items ();

        print COOKIE "delete", "\n";

    } else {
        &return_error (500, $error, "The cookie could not be set.");
    }

    &close_connection (COOKIE);
}
```

display_all_itemsサブルーチンはユーザのレスポンス要約をプリントします。

```
sub display_all_items
{
    local ($key_value, $key, $value);

    print "Content-type: text/html", "\n\n";
    print "<HTML>", "\n";
    print "<HEAD><TITLE>Summary</TITLE></HEAD>", "\n";
    print "<BODY>", "\n";
    print "<H1>Summary and Results</H1>", "\n";
    print "Here are the items/options that you selected:", "<HR>", "\n";

    while (<COOKIE>) {
        chop;
        last if (/^\.$/);

        $key_value = (split (/\s/, $_, 2))[1];
        ($key, $value) = split (/=/, $key_value);

        print "<B>", $key, " = ", $value, "</B>", "<BR>", "\n";
    }
```

whileループはサーバからの出力を読み込んで、キー値ペアを解析して表示します。

```
    foreach $key (keys %FORM) {
        next if ($key =~ /^Magic_Cookie$/);

        print "<B>", $key, " = ", $FORM{$key}, "</B>", "<BR>", "\n";
    }
        print "</BODY></HTML>", "\n";
}
```

この最終フォームからのキー値ペアも表示されます。これらはサーバには格納されません。

最後に、parse_form_dataサブルーチンは、クエリー文字列（GET）と標準入力（POST）の両方からキー値ペアを連結して、それを連想配列に格納します。

```
sub parse_form_data
{
    local (*FORM_DATA) = @_;

    local ($query_string, @key_value_pairs, $key_value, $key, $value);

    read (STDIN, $query_string, $ENV{'CONTENT_LENGTH'});

    if ($ENV{'QUERY_STRING'}) {
            $query_string = join("&", $query_string, $ENV{'QUERY_STRING'});
    }

    @key_value_pairs = split (/&/, $query_string);
```

```
        foreach $key_value (@key_value_pairs) {
            ($key, $value) = split (/=/, $key_value);
            $key   =~ tr/+/ /;
            $value =~ tr/+/ /;

            $key   =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
            $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;

            if (defined($FORM_DATA{$key})) {
                $FORM_DATA{$key} = join ("\0", $FORM_DATA{$key}, $value);
            } else {
                $FORM_DATA{$key} = $value;
            }
        }
    }
```

10.10 チャイルドプロセスのフォーク/生成

本章を終える前に、UNIXオペレーティングシステムの非常に強力な機能（コンカレントプロセス）について見ていきましょう。

ここで説明したクッキーサーバは、5つの接続が待ち行列に入ってきても、一度に1つしか接続を受け付けられず、1つ1つ順番にハンドルしていくことになります。変数に情報を格納するというサーバ操作のために、同時に複数の接続をハンドルするように設計することはできません。ここでは、その理由について見ていきましょう。

UNIXでは、プロセス（ペアレント）は、個別に指定したコードを実行する別のプロセス（チャイルド）を作成するための機能を持っています。これは、実行するのに膨大な時間がかかるプログラムにとっては非常に重宝します。例えば、いくつかの複雑な方程式を計算したり、大規模データベースを検索したり、大量のファイルを消去したりする必要があるCGIプログラムがあると、このタスクを実行するチャイルドプロセスを生成して、その間にペアレントがブラウザにコントロールを返すようにすることができます。この場合、ユーザはタスクが終了するまで待つ必要はなくなります。なぜなら、チャイルドプロセスはバックグラウンドで実行しているからです。それでは、簡単なCGIプログラムについて見ていきましょう。

```
#!/usr/local/bin/perl

$| = 1;
print "Content-type: text/plain", "\n\n";
print "We are about to create the child!", "\n";

if ($pid = fork) {
```

```
        print <<End_of_Parent;

I am the parent speaking. I have successfully created a child process.
The Process Identification Number (PID) of the child process is: $pid.

The child will be cleaning up all the files in the directory. It might
take a while, but you do not have to wait!

End_of_Parent

} else {
        close (STDOUT);

        system ("/usr/bin/rm", "-fr", "/tmp/CGI_test", "/var/tmp/CGI");
        exit(0);
}

print "I am the parent again! Now it is time to exit.", "\n";
print "My child process will work on its own! Good Bye!", "\n";

exit(0);
```

forkコマンドは実際にチャイルドプロセスを作成し、そのプロセスのPIDをペアレントに返し、ゼロの値をチャイルドに返します。この例では、コードの先頭ブロックはペアレントによって実行されます。一方、2番めのブロックはチャイルドによって実行されます。ここで注意すべきことは、チャイルドプロセスは、ペアレントで利用可能なすべての変数とサブルーチンのコピーを獲得するという点です。ただし、チャイルドプロセスが少しでも修正を行うと、これらのコピーは終了するときに破棄され、ペアレントプロセスには影響を与えません。

クッキーサーバが複数の接続をハンドルできない大きな理由は以下のとおりです。ここでは問題点は2つあります。まず1つは、複数の接続がサポートされていない点です。CGIプログラムがサーバに接続すると、サーバはプログラムからのリクエストをハンドルします。したがって、プログラムがその接続を中断するまでは、ほかの接続を受け付けることはできません。複数接続を可能にする唯一の方法は、各接続をハンドルするための新しいプロセスが登場するように、接続ごとにプロセスをフォークすることです。

これは2つめの問題を引き起こします。各接続をハンドルするための別々のチャイルドプロセスがあると、各プロセスは（ペアレントのデータコピーが入っている）自分の変数を持つようになります。チャイルドプロセスが新しいデータを修正したり（変数に）格納したりすると、そのデータはプロセスが終了したときに破棄されてしまい、そのデータをペアレントに戻す方法がなくなってしまいます。1つのサーバが一度に1つの接続データしか維持管理しないのは、このような理由からです。

UNIXコマンドを実行するときに使用してきたsystemコマンドは、以下のように実装されています。

```
unless (fork) {
    exec ("command");
}

wait;
```

これは次と同じです。

```
system ("command");
```

基本的には、フォークからの戻り値がゼロのときだけunlessブロックが実行されますが、このチャイルドプロセスは指定したコマンドを実行し、一方、ペアレントはこれを待ってから終了します。複数の接続を同時にハンドルするサーバを実装する方法を以下に示します（ただし、このアプローチはここで紹介したクッキーサーバでは動作しません）。

```
$SIG{'CHLD'} = "wait_for_child_to_die";

while (1) {
    ( ($ip_name, $ip_address) = &accept_connection (COOKIE, SOCKET) )
        || die "Could not accept connection.", "\n";

    if (fork) {
        #
        # ペアレントプロセス（ほとんど何もしない）
        #
    } else {
        #
        # チャイルドプロセス（ほとんど全ての処理を行う）
        #
    }

    &close_connection (COOKIE);
}

sub wait_for_child_to_die
{
    wait;
}
```

1つ重要なことがあります。ペアレントがチャイルドプロセスを待たずに消滅すると、ゾンビプロセスがシステムに残ります。

CGIプログラミング

1996年11月15日　初版第1刷発行
1997年 1 月10日　初版第2刷発行

著　　者　Shishir Gundavaram（シシャ・ガンダヴァラム）
監 訳 者　田辺 茂也（たなべ しげや）
訳　　者　株式会社 エディックス
発行・編集人　河村 由美子
編集・制作　株式会社 スペック
印刷・製本　三美印刷株式会社
発 行 所　株式会社 オライリー・ジャパン
〒160　東京都新宿区三栄町12番地　清重ビル2F
TEL （03）3356-5227
FAX （03）3356-5261
発 売 元　株式会社 オーム社
〒101　東京都千代田区神田錦町3-1
TEL （03）3233-0641（代表）
FAX （03）3293-6224

Printed in Japan （ISBN4-900900-13-3）　定価 4,996 円（税込）
落丁、乱丁の際はお取り替えいたします